Automatic Construction of Synthetic Musical Instruments and Performers

Ning Hu

CMU-CS-13-115

July 2013

School of Computer Science
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Roger B. Dannenberg, Chair
Richard M. Stern
Daniel D. Sleator
David Wessel, U.C. Berkeley

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Keywords: Computer Music, Musical Instrument Synthesis, Machine Learning

Dedicated to my parents, whose unconditional love supports me through every step of the way.

Abstract

This thesis describes an approach to create high-quality music synthesis by automatically constructing an instrument model and a performance model; the latter module generates control signals from score input and drives the former module to produce synthetic instrumental sounds. By designing and applying appropriate machine learning techniques as well as domain knowledge, the instrument model and the performance model are constructed from acoustic examples and their corresponding scores for a musical instrument. The automated model is able to synthesize realistic instrumental performances from scores.

Acknowledgments

The thesis would not have been possible without my advisor, Roger B. Dannenberg. He repeatedly encouraged me to finish my thesis and never gave up on me, even long after I left Carnegie Mellon and started working on completely different things in the industry.

This research topic is greatly inspired by Roger B. Dannenberg and Istvan Derenyi's outstanding work on designing and developing the combined spectral interpolation synthesis technique. As a trumpet musician, Roger B. Dannenberg also performed and helped record the acoustic examples for initial experiments. Other acoustic samples used by this research are the bassoon recordings performed by Mark Dalrymple.

I should also especially thank Guanfeng Li for his tremendous help and moral support during the thesis writing process.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

For over 50 years, people have been trying to understand how acoustic instruments, such as the trumpet, sound so beautiful, and how to simulate them with the computer. This thesis research is an attempt to find an elegant and working solution for such a grand question. The goal is to create an automated system that can model musical instruments by learning from acoustic recordings. Ultimately it should be able to synthesize high-quality instrumental performances given only the score as input.

The two most basic modules of the proposed framework are the instrument model and the performance model. The function of the instrument model is similar to that of ordinary music synthesis, that is, to generate synthetic sound given control signals as input; the performance model is in charge of converting the given digital score into these control signals, which is essentially the driver of the instrument model.

This research emphasizes designing and implementing a system framework that can automatically construct an instrument model and a performance model not only by deploying necessary domain knowledge, but more importantly, by intelligently learning from acoustic examples. The raw input of the system consists of acoustic recordings and their corresponding scores, but these cannot be used without careful annotation and labeling, which could be extremely labor intensive. Thus, automated data pre-processing is also a crucial part of the system. An automated

system with note onset identification and score alignment functionalities generates detailed labels that are used in constructing the instrument and performance models. Various machine learning techniques are applied throughout the framework to automate the construction process.

The current infrastructure is especially targeted at modeling wind instruments, and the first instrument to be modeled is the $B\flat$ trumpet. There are two main reasons for that.

First, most current commercialized synthesizers fail to synthesize realistic sounds of wind instruments. The problem lies in the conflict between the basic structure of synthesizers and the working mechanisms of wind instruments. On the one hand, wind instruments are controlled continuously by a human player. This continuous control drives the sound production. On the other hand, synthesizers (mostly sampling-based) are based on single, isolated notes, and do not offer a wide range of control over the spectrum, attacks, and envelopes. They can synthesize a single note very well, but when they are used for synthesizing a phrase or a passage, listeners will immediately notice problems.

Second, previous research by Dannenberg and Derenyi [10] shows a similar scheme is capable of producing convincing trumpet sounds; it is natural to start from something known to be working.

For music style, this thesis research mainly focuses on synthesizing classical music, as the playing style is purer, more faithful to the score, and has fewer articulation effects that require additional tunings of the proposed system. In many non-classical music-playing techniques, inharmonic and transient sounds are significant. However, modeling the noise (or the non-harmonic part) of musical instruments is not the primary purpose of this research. Still the system should be flexible enough that it can be extended to synthesize other types of music in the future.

In the following chapters, we will first state the contributions of the thesis and give an overview of the related work; then we will describe the system infrastructure and fundamental processes and further explain each important module in detail, including the audio alignment and segmentation for data pre-processing, the instrument model, and the performance model; before the conclusion chapter, we will present an effort in extending the system to model other

types of wind instruments, using the bassoon as a specific example.

1.1 Thesis Statement

The goal of this thesis research is to demonstrate that, by designing and applying appropriate machine learning techniques, we can automatically create a high-quality musical instrument synthesizer from performance examples and corresponding scores.

Several major branches in music synthesis all suffer from the problem of control. Sampling and additive synthesis [48] both have problems due to the fact that they are focusing on notes rather than either the production of sound or control mechanisms. Physical models [51] promise to solve the problems of synthesis, but direct models of acoustic instruments are very difficult to build, they are highly specific to individual instruments, and accurate models should require equally accurate gestural control of many parameters to achieve musical results. The proposed method avoids the problem of note-oriented synthesis, bypasses the difficulties of building physical models, and simplifies the problem of control with spectral models and effective machine learning techniques. It provides a new approach to automatically model different musical instruments, which makes it interesting to the world of music synthesis and also to machine learning applications.

Chapter 2

Related Work

This thesis work builds on much previous work in computer music, some of which dates back several decades. In this chapter, we outline work related to this thesis research. We first enumerate the fundamental approaches that laid the ground work for musical instrument synthesis, then we talk about several advanced synthesis techniques or applications that are built upon these basic synthesis approaches. Furthermore, many music synthesis techniques owe much to early speech synthesis research, as many issues of synthesis and control are common to both music and speech synthesis. Therefore we describe here different types and algorithms of speech synthesis, and give a brief account of their historical evolution as well. Last we will introduce the recent progress on singing voice synthesis, which is an interesting interdisciplinary topic that can be viewed as a fusion of speech and music synthesis. All the related work forms the inspiration and foundation of this thesis research.

2.1 Music Synthesis

2.1.1 Basic types of synthesis

The techniques laid out here in this subsection are the theoretical fundamentals for music synthesis. Advanced approaches and applications in later subsections are either combinations of these basic techniques, and/or extensions with specifics unique to a class of music instruments or to a domain.

2.1.1.1 Sampling synthesis

Sampling synthesis is a form of audio synthesis that generates the synthesized sound using, as the name suggested, sampled sounds. These sampled sounds are acoustic recordings of an instrument playing a range of pitches (and sometimes different articulations, dynamics, etc.).

The principal advantage of sampling synthesis over other types of sound synthesis, such as FM synthesis and additive synthesis, is that subtle sound qualities of acoustic instruments are easily captured through recordings. Computation is relatively inexpensive, consisting mostly of resampling for pitch control.

However, sampling synthesis also has obvious drawbacks, one being the hefty consumption on memory, and the other prohibitively expensive development for covering the full range of instrument sounds.

To reduce the memory consumption, especially in the early days when computer memory was quite expensive, the samples had to be as short and as few as possible. A common memory-saving technique is to loop a steady-state portion of the sample, and then use an amplitude envelope curve to make the sound fade away. Also an amplifying stage is used to translate key velocity into gain, so that harder playing results in louder playback. A further refinement could be modulating the attack time of the instrument using key velocity, leading to a faster attack for loud notes [29].

The second disadvantage is the inefficiency for rendering expressive and realistic instrument

sound. Pianos are relatively easy to model, as they play only 88 pitches with velocity as the single control dimension (pedals aside) of any key. But for many instrument families, it is completely impractical to capture the complete range of sound. That is why we have yet to see note-based sampling synthesis that can produce expressive and convincing sounds of any bowed string instrument or any wind instruments.

As the development of computing hardware follows Moore's law, memory becomes cheaper and cheaper, and makes the memory consumption aspect of sampling synthesis less of an issue. Now it is quite common to see a sampling synthesis model that consumes memory of 1 gigabyte or more. Inexpensive memory means more samples can be stored and retrieved efficiently in real time. While early sampling synthesizers may have stored one sample for each octave and transposed samples to produce a full set of pitches, modern samplers can take advantage of cheap memory by storing a sample for every semitone of pitch. This technique provides a more natural progression from the lower to the higher registers so that lower notes do not sound dull, and higher notes do not sound unnaturally bright. It is also possible to reflect volume and timbre changes by sampling the same note at several different levels of intensity. As an example, one can make 3 samples per key for a piano: soft, medium and loud. All other volume levels in between can be created by amplifying and blending the samples. Multi-samples across both pitch and force of playing are usually preferred in order to make sampled instruments sufficiently expressive [69].

Thus the quality of a sampling synthesizer is mostly determined by the quality of the sampled sounds and the sampling coverage of different playing conditions of the instrument. Since any sound an instrument makes can be recorded, there is no fundamental lack of generality in sampling synthesis. Sampling synthesis is by far the most commonly used synthesis technique, and the majority of the commercial synthesizers today use this method with various refinements.

2.1.1.2 FM synthesis

Frequency Modulation (or FM) synthesis is a simple and powerful method for creating and controlling complex spectra, discovered by John Chowning of Stanford University around 1973 [8]. While experimenting with different types of vibrato at Stanford University in the mid-60's, Chowning found that when the frequency of the modulating signal increased beyond a certain point, the vibrato effect disappeared from the modulated tone, and a complex new tone replaced the original. In its simplest form, FM involves a sine wave carrier whose instantaneous frequency is varied, i.e. modulated, according to another sine wave (the so-called modulator).

The output waveform is given by

$$output = a_c \cos(w_c t + a_m \cos(w_m t))$$

where a_c (a_m) and w_c (w_m) are the maximum amplitude and frequency of the carrier (modulator). As we can see from the formula, in FM, the frequency of the carrier (or more precisely, the phase) is swept up and down by the modulator. Consequently, a series of side bands are generated whose frequencies are given by:

$$w_{sb} = w_c \pm n \times w_m$$

where n is an integer.

The amplitudes of the side bands, i.e., the shape of the resulting spectrum, is determined by a new concept, "modulation index" or simply β as given in:

$$\beta = \frac{\Delta w_c}{w_m}$$

Since the numerator of this expression is the change in the carrier frequency, which is proportional to the amplitude of the modulator, it means for any given modulator frequency, the amplitude of the Modulator determines the amplitude of each of the components of the spectrum

of the output signal.

FM synthesis can create both harmonic and inharmonic sounds. To produce harmonic sounds, the modulating signal must have a harmonic relationship to the original carrier signal, that is, the modulating frequency must be an integer or small integer ratio multiple of that of the carrier. To produce bell-like dissonant or percussive sounds, modulators with frequencies that are non-integer multiples of the carrier signal must be used.

The beauty of FM synthesis lies in the ability to produce complex spectra with great simplicity. This technique, although not a physical model for natural sound, is shown to be a very powerful perceptual model for many musical instruments ranging from brass and woodwinds to strings. However, what is lacking is an automatic and general analysis algorithm to obtain FM parameters from instruments sounds. Furthermore, it is hard for one to gain intuition as to how these parameters will affect the resulting sounds.

2.1.1.3 Additive synthesis

Additive synthesis refers to a sound synthesis technique that creates timbre by adding sine waves together [67]. In the light of Fourier Transform theory, the timbre of musical instruments can be considered to consist of multiple harmonic or inharmonic partials or overtones. Each partial is a sine wave of a different frequency and amplitude that swells and decays over time. Additive synthesis generates sound by adding the output of multiple sine wave generators, or digitally it may also be implemented using pre-computed wavetables or inverse Fast Fourier transforms.

Harmonic additive synthesis is closely related to the concept of a Fourier series. Thus, additive synthesis is also called "Fourier synthesis". This Fourier series expresses a periodic function as the sum of sinusoidal functions with frequencies equal to integer multiples of a common fundamental frequency. These sinusoids are called harmonics, overtones, or generally, partials. In general, a Fourier series contains an infinite number of sinusoidal components, with no upper

limit to the frequency of the sinusoidal functions and includes a DC component. In additive synthesis, however, frequencies outside of the humanly audible range can be omitted. Even within the human audible range, partials with high frequencies, which express a delicate part of the sound, can be omitted to reduce the computation. As a result, only a finite number of sinusoidal terms are modeled in additive synthesis. Using inharmonic partials, additive synthesis can also generate inharmonic sounds such as those of bells.

It is common in practice to analyze the frequency components of a recorded sound to create a sum of sinusoids representation. This representation can then be re-synthesized using additive synthesis. One method of decomposing a sound into time varying sinusoidal partials is the Fourier Transform-based McAulay-Quatieri Analysis [39]. By modifying the sum of sinusoids representation, complex timbre features can be achieved prior to re-synthesis. For example, a harmonic sound could be restructured to sound inharmonic, and vice versa. Additive analysis/re-synthesis has been employed in a number of techniques including Sinusoidal Modeling [31], Spectral Modeling Synthesis (SMS) [58], and the Reassigned Bandwidth-Enhanced Additive Sound Model [17]. For instance, Irizarry carried out a statistical analysis to model the two elements of the sound as an additive sinusoidal part, and a residual energy part, which resulted in an even more complex timbre of the sound [30].

Additive synthesis offers powerful expressive control over the sound since the input parameters give a very accurate definition to the time-varying spectrum. Furthermore, these parameters lend themselves to a perceptual intuitive interpretation of the nature of the sound. Additive synthesis has proven to be highly useful as a perceptual description of acoustic instruments and offers easy timbre characterization and manipulation. However, it requires a lot of data since the frequency and amplitudes of each partial have to be stored. This causes several problems. In particular, control is very complex, so in practice it is difficult to build additive instrument models controlled by intuitive parameters such as pitch, duration, articulation, etc. In addition, the production of the sound is expensive because one tone requires many oscillators, each with independent amplitude and frequency control.

2.1.1.4 Concatenative synthesis

Concatenative synthesis is a technique for synthesizing sounds by concatenating short segments of recorded sound (called units). The duration of the units is not strictly defined and may vary according to the implementation, roughly in the range of 10 milliseconds up to 1 second [68]. Sampling synthesis can be seen as a special case of concatenative synthesis where units are notes, and units can be computationally manipulated at least to adjust pitch and duration. Concatenative synthesis is often used in speech synthesis and music synthesis to generate user-specified sequences of sound from a database built from recordings of other sequences.

Generally, concatenative synthesis produces the most natural-sounding synthesized sound since it comes from stringing together short pieces of natural sounds. However, differences between variations and transitions from one piece to the next in natural sound and the nature of the automated techniques for segmenting the waveforms sometimes result in audible glitches in the output. The reason that concatenative synthesis was first and heavily explored in speech synthesis is that intuitively, human speech can be naturally broken down to pieces and one can reorganize those pieces into different sequences to express different meanings.

The unit in all kinds of speech synthesis, however, can differ vastly. In unit selection synthesis, each recorded utterance is segmented into some or all of the following: individual phones, diphones, half-phones, syllables, morphemes, words, phrases, and sentences. Typically, the division into segments is done using a specially modified speech recognizer with some manual correction afterward, using visual representations such as the waveform and spectrogram [5]. An index of the units in the speech database is then created based on the segmentation and acoustic parameters such as the fundamental frequency (pitch), duration, position in the syllable, and neighboring phones. At run time, the desired target utterance is created by selecting the best string of candidate units from the database using a weighted decision tree. In diphone synthesis, the unit database contains all the diphones (sound-to-sound transitions) occurring in a language. At runtime, the target prosody of a sentence is superimposed on these minimal units by means of

digital signal processing techniques such as linear predictive coding, PSOLA [44] or MBROLA [16]. Diphone synthesis suffers from the sonic glitches of concatenative synthesis and a robot-like sound often results. Therefore, its use in commercial applications is declining.

Concatenative synthesis for music started to develop in the 2000s. The basic techniques are similar to those for speech, although with differences due to the differing nature of speech and music. For example, the segmentation is not into phonetic units but often into subunits of musical notes or events. In his survey paper [56], Schwarz enumerated current research on concatenative synthesis and identified multiple urgent questions on future work. Zils and Pachet [71] proposed Musical Mosaicing to generate sequences of sound samples automatically by specifying only high-level property of the sequence.

Concatenative synthesis faces the same dilemma as sampling based synthesis does. Basically, it can reproduce all the sounds in great detail as long as a large pre-captured database is given. However, it is impossible to record all sound segments of an instrument and trading algorithmic transformations for memory space will result in perceptual glitches.

2.1.2 Advanced synthesis approaches and applications

2.1.2.1 Synful

Synful Orchestra is the first series of commercial products that try to combine the realistic sound quality of sampling with the expressivity of a real performer. Synful Orchestra, at its core, is a new synthesis technology called Reconstructive Phrase Modeling (RPM) [35], which puts great emphasis on capturing the dynamics of note transitions. RPM is an analysis-synthesis system that is related to two synthesis technologies described in earlier sections: the first is a form of additive synthesis in which sounds are generated as a sum of sinusoidal waves with one base frequency and its harmonics; the other related technology is concatenative synthesis. RPM searches a database of idiomatic instrumental phrases and combines modified fragments of these phrases to form a new phrase.

The input to RPM is a MIDI-like control stream, which may be generated automatically from a composer's score or can originate from a controller such as a keyboard in real time. The output of RPM can be seen as rapid fluctuation in pitch and loudness signal superimposed on a slowly varying one. The slowly varying pitch and loudness controls, which determine the basic timbre of an instrument, can be derived directly from the input MIDI control stream. The rapid fluctuation in RPM, which gives the synthesized sound richness and realism, is searched and selected from a database with some morphing such as stretch, shift and slide. To generate such a database, solo sessions of complete idiomatic musical passages that represent all kinds of articulations and phrasing, such as detached, slurred, portamento, sharp attacks, soft attacks, etc., are recorded. A note-oriented trend removal is performed upon the recorded sessions; the leftovers are stored as the rapidly varying fluctuations. These phrases are labeled by descriptor information that identifies the pitches, length of notes, intensity of notes and the type of note transitions. In the synthesis phase, RPM parses the input MIDI stream to locate musical phrases, usually consisting of two to eight notes. When a phrase is identified, a search is performed in the database to find a closest match on pitches, note duration, note transition, etc. The selected phrase is then morphed to best match the phrase in the MIDI sequence. The morphing operations include stretching or compressing the notes in time, shifting notes in pitch and intensity, sliding notes in time and modifying intensity and speed of vibrato, etc.

Separating slowly varying components of the sound from rapid ones yields a few advantages. Since the slowly varying components are responsible for the majority of the energy in the output audio signal and they are derived directly from the MIDI stream, RPM is almost immediately responsive to the performer. At the same time, since the processing unit is a musical phrase, the discontinuity in RPM is greatly masked since it mainly comes from the morphing of the rapidly varying components. Furthermore, the rapidly varying components generally contain variations related to vibrato. It is possible to provide near real-time control of vibrato by varying the readout rate of the rapidly varying components. However, when the input MIDI is in real-time, and no look-ahead is allowed, the quality may suffer as note transition information such

as slurs is not known to the system. Thus to ensure the synthesis quality, a small time lag, e.g. 50ms, is introduced to the system when handling real-time MIDI.

RPM achieves musical expressivity through a combination of additive synthesis and a phrase-oriented catenative synthesis. The realism and richness of the natural sound played by musical instruments are preserved by a database of rapidly varying components derived from original recordings. By working on complete phrases, RPM emphasizes transitions between notes and phrasing gestures that span several notes.

2.1.2.2 Dannenberg's early work

Dannenberg and colleagues have conducted extensive research on the synthesis of wind instruments [10, 11], especially on that of trumpets. It is observed that for wind instrument synthesis, it is not adequate to simply generate spectra for individual notes and string all together to produce a musical piece. Since the performer exerts continuous control over amplitude, frequency and other parameters, transitions between notes are important, and the details of a tone for a note are affected by its context. To address these problems, they came up with a two-stage synthesis system with a performance model to generate appropriate control signals from musical scores and an instrument model to convert these signals to corresponding time-varying spectra.

The key to the instrument model is the realization that the time-varying spectrum is almost entirely determined by relatively simple modulation sources such as amplitude and frequency. It is true even when the amplitude fluctuates rapidly. These signals can be easily extracted from musical scores and make intuitive sense to music performers.

The instrument model is fundamentally an additive synthesis, where the primary modulation sources are the current RMS amplitude and the fundamental frequency. At every time-point, a stored database of spectra indexed by amplitude and frequency is consulted and the output spectrum is generated. For the inharmonic attack transitions where additive synthesis would not work, carefully made splices transition from the recorded attacks to synthesized tones to generate

natural-sounding attacks.

To generate amplitude and frequency controls to feed the instrument model, the envelope of the trumpet was studied statistically. The analysis revealed that the envelope changes systematically according to the melodic context, dynamic level and articulation. Inspired by the study, a performance model of trumpet envelopes was created in which properties of a symbolic musical score determine a number of parameters of the envelope. The resulting envelopes can be seen as breath envelopes modulated by tongue envelopes. Breath envelopes are built from different portions of an actual envelope stretched to appropriate duration. A center portion will result in a fairly flat envelope; an earlier or a later portion will result in rising or falling envelopes respectively. For tongue envelopes, analytic functions composed from sines and exponentials are used, and functions are fitted to actual data in order to derive a set of typical parameters. The resulting envelopes are then fed to the instrument model described above. The generated trumpet phrases were realistic, including proper dynamics, spectral variation, tongued attacks and slurred note transitions.

Subsequent efforts were made to automate the collection of data for building instrument models [12]. In this work, recordings were automatically segmented, but the results were not very satisfactory, with only 65% of notes segmented and analyzed without problems.

2.1.2.3 Wessel's early work

Analysis-synthesis has for the most part been used by composers to reproduce musical phrases. Given a large sample of original sound, they were able to generate almost all the sounds a musical instrument can play. However, the preservation of the order of sound evolution those methods entails greatly constrains the musicians from constructing new phrases. Wessel et al. explored the possibility of obtaining an instrument model by analyzing the control signals and their corresponding output spectral details [66], so that they can play a new melodic figure by supplying the model with only control signals such as pitch, loudness and brightness.

Towards that goal, they treated the instrument model as a black box and investigated two approaches: a multi-layer neural network supervised learning model and a memory-based model that reorganizes the spectral frames in a matrix indexed by the control signals. In the neural network model, they used a back propagation learning method to fit the functions to get the parameters; the inputs accept the pitches and loudness functions while the outputs produce the frequencies and the amplitudes of the sinusoidal waves for the additive model. The training data is exhausted and discarded. For the memory-based model, however, the training data is not fitted once and discarded, but kept for references to generate output by combining local exemplars determined by the input control signals. For each input, distances to all the data points are computed and k-nearest neighbors are selected. A weight vector is calculated for those k-selected neighbors using a Gaussian kernel.

Experiments were carried out using the aforementioned two models; the authors reached the conclusion that both models produce acceptable results in a real time context. The memory-base model though very memory intensive, seems to be more flexible, easier to modify, and more adaptive for creative use. The neural network model, on the other hand, is very compact and appears to generalize well. In general, the neural network model provided a smoother sounding result than the memory-base model.

2.1.2.4 Wind instrument synthesis toolbox

Wind instrument synthesis toolbox is a set of software tools for the automatic generation of synthesized audio files accompanied with labels that describe the temporal evolution of the amplitude and frequency of each one of the partials present [52]. It is based on a similar observation with Dannenberg that wind instruments bear some common characteristics. For example, all of them consist of a tube resonator and some kind of acoustical excitation that makes the air within the tube vibrate and resonate at a certain fundamental frequency. The vibration and resonance produce a harmonic spectrum whose components tend to decrease in amplitude with

frequency. The waveform amplitude evolution of wind instruments can be usually divided into attack, steady-state or sustain and release. During the attack and release, the envelope can be roughly approximated with an exponential curve. However, some notes can have a more complex behavior, e.g. a pronounced attack followed by a short decay, sustain and then release. As for the spectral behavior, wind instruments also exhibit broad similarities. The harmonics from the fundamental frequency seem to appear one after another and fade out in the opposite way. Therefore, the sound perceptually becomes brighter gradually during the attack, until it reaches its maximum in the sustain and gets darker during the release. Additionally, brightness also changes with dynamics in the same way, that is, the relative amplitudes of higher harmonics increase with intensity.

Motivated by the aforementioned observations, Horner and Ayers [22] proposed a general synthesis system for wind instruments, which applies additive synthesis to an analysis model with some simplifications. The first simplification is that contiguous partials are grouped in disjoint sets in order to control their temporal amplitude evolution. The disjoint sets are approximately corresponding to the division of the frequency range by critical bands, that is, the first group has only the fundamental frequency component, the second group the second and third partials, the third group from fourth to seventh component and the fourth group has all the remaining harmonics. In this way, the amplitude evolution of each partial, rather than being independent, is changing in the same way. Another simplification consists in selecting a single representative spectrum of the sustain part of a note. This spectrum is dynamically modified by changing the amplitude of each group in such a way to produce the behavior of the wind instruments described previously. The selection of the representative spectrum can be performed by means of optimization techniques, or just by picking a spectrum with average brightness. The latter has been proven to have negligible impact on the timbre perception while reducing much computation. Using a less precise model of the time evolution of the partials further reduces the amount of information that is retained from the analysis, yet with little performance sacrifice.

In order to make the synthesized sound more natural, the synthesis model also includes a

dynamic vibrato as performers typically modify their vibrato during the course of a note. However, the synthetic nature of the sounds produced is still recognizable during aural evaluation. This is mainly due to the lack of natural articulations and discrete timbre variations along the register. An additional deficiency of the model is that timbre changes produced by different dynamics are not taken into account. Therefore, increasing the number of spectral reference notes, including also different dynamics, would seem to offer important improvement. To this end, a collection of steady-state spectrum estimates was obtained from the McGill University Master Samples (MUMS). For notes played in different dynamics, steady-state spectrum analysis was implemented on Iowa Musical Instrument Samples (MIS). Aural tests showed substantial improvement over the results of the synthesis performed with the original model data.

2.2 Speech Synthesis

Speech is one of the primary means for people to communicate. Speech synthesis (sometimes referred to as Text To Speech (TTS)) has been under development for decades. Of these many synthesis technics, we can roughly categorize them into three generations: the first generation often requires detailed rules and low-level description of what is to be uttered, the representative approaches are formant synthesis and articulatory synthesis. The second generation uses a data driven approach, and the memory requirement to host prerecorded voice samples is high. Concatenative synthesis dominates this generation of speech synthesis. The third generation employs statistical, machine learning techniques to infer the prosodic information for new utterances from a small training set, such as sinusoidal synthesis and HMM based synthesis. Singing voice synthesis is a synthesis bearing special interest from both music sound synthesis and speech synthesis. We devote a section for it here.

2.2.1 Formant synthesis and articulatory synthesis

Formant synthesis and articulatory speech synthesis are two representative approaches in the first speech generation epoch. Formant synthesis is often called synthesis by rule. The basic assumption is to model the vocal tract transfer function by simulating formant frequencies and amplitudes. It makes use of the acoustic-tube model that mimics the human speech apparatus. A sound is generated from a source, with a periodic signal for voiced sounds and white noise for unvoiced sounds. The basic source signal is then fed into the vocal-tract model to produce a speech pressure waveform. Formant synthesis, though very reliable, generates artificial and robotic-sounding speech. It requires a small memory footprint and reasonable computation power. Likewise, articulatory speech synthesis uses a mechanical and acoustic model of speech production to produce speech. It transforms a vector of anatomic or physiologic parameters into a speech signal with predefined acoustic properties based on mathematical models of how the human vocal tract structure processes speech. Articulatory speech synthesis can produce speech with very complex patterns; however, it requires intensive computation, and it is very hard and counterintuitive to generate the control vectors.

2.2.2 Concatenative synthesis

Concatenative synthesis generates speech by combining splices of pre-recorded natural speech from a database, and has dominated speech synthesis since the early 80's. There are two major issues in concatenative synthesis, the units consisting of the database and algorithms to select appropriate units to string them to form the final speech. The units can be of some or all of the following: individual phones, diphones, triphones, syllables, morphemes, words, phrases and sentences. For example, a diphone synthesis database contains all possible diphones in a language. We have previously explained how concatenative synthesis works in Section 2.1.1.4, when we introduced synthesis approaches for music. Since concatenative synthesis works on recorded natural speech, the sounds it produces really sound like human utterances, especially

in domain based synthesis, where the output utterances are limited and restrained in specific domains such as flight announcements or automatic weather reports. However, while the discontinuity between units may be alleviated by signal processing methods, it is still perceivable and it is very hard to generate output with different prosodic features such as emotions from a limited database.

2.2.3 Sinusoidal synthesis and HMM based synthesis

More recent advances in speech synthesis include sinusoidal synthesis and Hidden Markov Model (HMM) synthesis.

The human voice can be modeled as a set of harmonics of an estimated fundamental frequency and some random noise. The first stage of sinusoidal synthesis is to classify the frames into voiced and unvoiced portion and determine the parameters of the harmonic and noise components and the relative strength of their contribution to the frame [21]. The fundamental parameters such as amplitudes, frequencies and phases of the harmonics can be changed by keeping the same spectral envelope to get other effects, such as changing the whole sentence from a male speaker to a female one.

HMM speech synthesis is a statistical machine learning approach to simulate real life stochastic processes of human utterance [70]. In HMM based synthesis, the speech parameters such as frequency spectrum, fundamental frequency and duration are statistically modeled and speech is generated by using an HMM based on maximum likelihood criteria. An HMM is a collection of states connected by transitions. Each transition carries two sets of probabilities: a transition probability, which defines the probability of that transition being taken and an output probability density function, which provides the conditional probability of emitting certain outputs given that the transition is taken. In the synthesis stage, an arbitrarily given text to be synthesized is converted to a context-dependent label sequence and a sentence HMM is constructed by concatenating context dependent HMMs according to the label sequence. The state duration can also be

estimated by providing state duration probability distributions at the training stage.

HMM synthesis provides a means by which to train the specification-to-parameter module automatically, thus avoiding laborious hand-written rules. The trained models produce high quality synthesis speech and have the advantage of being compact and amenable to modification for voice alteration and other purposes. HMM synthesis seems to be able to generate very natural sounding speech with more complex prosody.

2.2.4 Singing voice synthesis

Singing voice synthesis can be seen as a hybrid of speech synthesis and music synthesis, the lyrics part of the synthesis obviously bears some resemblance of speech synthesis while special processes such as fundamental frequency handling and vibrato generation are key to music synthesis. Among recent singing voice synthesis systems, two methods stand out on the frontier research of the field. One is a corpus based HMM approach, building the singing voice from scratch and a physical model based approach, converting a speech (reading of the lyrics) to a song.

2.2.4.1 HMM based approach

HMM-based singing voice synthesis was built upon the HMM-based speech synthesis described above [70]. Saino et al. [54] modeled musical information such as lyrics, tones, durations of the phonemes simultaneously in a unified framework of the context-dependent HMMs. The system is composed of a training and a synthesis part. The spectrum, excitation, and vibrato are extracted from a singing voice database in the training part and they are then modeled with context-dependent HMMs. Context-dependent models of state durations are also estimated. To produce a singing voice of an arbitrarily given musical score including the lyrics, first the score is converted to a context-dependent label sequence. Second, according to the label sequence, an HMM corresponding to the song is constructed by concatenating the context-dependent HMMs.

The state durations of the song HMM are then determined with respect to the state duration models. Fourth, the spectrum, excitation, and vibrato parameters of the speech parameters are generated. As the last step, the singing voice is synthesized directly from the generated spectrum, excitation, and vibrato parameters by using a Mel Log Spectrum Approximation (MLSA) filter.

The quality of synthesized singing voices strongly depends on training data because HMM-based singing voice synthesis systems are corpus-based. It is very costly to build a database that covers all possible contextual factors such as keys, lyrics, dynamics, note position and durations, pitch, etc. Pitch should be particularly covered very well since it has a great impact on the subjective quality of synthesized singing voice. Oura addressed the data sparseness problem with his colleagues by using pitch-shifted pseudo-data [47]. In addition, the pitch normalization was done in conjunction with the HMM parameter estimation, other parameters such as spectrum, excitation, vibrato, or state duration can also be estimated with the shifted pitch. The extension can produce synthesized voice with a relatively small singing voice database.

2.2.4.2 Speech to singing synthesis

A speech to singing synthesis method was proposed by Saitou et al. [63], which converts a speaking voice reading the lyrics of a song to a singing voice given its musical score. It was based on the observations of three distinct acoustic features unique to singing voices: the fundamental frequency F0, phoneme duration and spectrum. Consequently, three models were proposed to handle the corresponding three features in the synthesized voice.

The targeted F0 is derived directly from the musical score of the song. The global F0 contour is determined by the musical notes and local F0 varies according to F0 fluctuations. F0 fluctuations include overshoot at beginning of a note, preparation at the end of a note and vibrato in between. There are fine fluctuations modeled as noise.

Because the duration of each phoneme of the speaking voice is different from that of the singing voice, it should be lengthened or shortened according to the duration of the correspond-

ing musical note. Note that each phoneme of the speaking voice is manually segmented and associated with a musical note in the score in advance. The duration of each phoneme is determined by the kind of musical note and the given local tempo.

In previous work [45, 61], two spectral characteristics unique to singing voices were reported. Sundberg [61] showed that the spectral envelope of a singing voice has a remarkable peak called the singing formant near 3 kHz. Oncley [45] reported that the formant amplitude of a singing voice is modulated in synchronization with the frequency modulation of each vibrato paired in the F0 contour. The spectral envelope of the speaking voice is modified by controlling these two features to obtain the final singing voice spectral envelope.

The extracted parameters are then fed to a speech synthesizer to get the singing voice. Experiments showed that the naturalness of the synthesized voice is close to that of actual singing voices and F0 fluctuations are more dominant acoustic cues than the spectral characteristics in the perception of singing voice. Furthermore, since the F0 and spectral envelopes can be independently controlled, acoustic features that affect the singer's individuality and style can be expressed by the proposed system.

Chapter 3

System Architecture

The core system of this thesis research is a synthesis framework, which renders an audio performance from a symbolic score. The synthesis framework itself is rather simple and straightforward, while the construction process takes several automated steps.

3.1 Synthesis Framework

As shown in Figure 3.1, the synthesis system can be naturally divided into two sequential modules. The performance model gets the score input and produces a set of control signals as output. The instrument model accepts the control signals generated from the performance model as input, and synthesizes output audio. Here control signals are continuous and play a key role as an intermediate representation in the rendering process.

A more detailed diagram is shown in Figure 3.2. The score is represented in MIDI as the input, while the synthesized sound is the audio output. Note context, including its position within a slur and relationship among adjacent notes, is extracted from MIDI. Although we use MIDI for convenience, we can alternatively extract context information from another representation of the score, MusicXML. A sequence of note context descriptors, each represented by a set of parameters, is the input of the performance model. The performance model consists of two sub-models

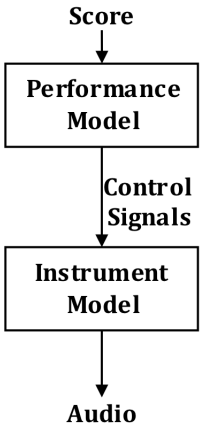


Figure 3.1: The synthesis framework

the amplitude model that generates amplitude envelopes using a learned decision tree, and a frequency model that synthesizes frequency envelopes with a set of hand-tuned functions. The generated envelopes of amplitude and frequency are the control signal inputs of the instrument model, which also consists of two sub-models, the harmonic model and the attack model. The outputs of these two models are spliced together to generate the final audio output.

3.2 Synthesis Construction Process

In order to build such a synthesis framework in a systematic way, we need to accomplish the following processes.

- Generate labeled data by aligning acoustic recordings to their corresponding scores and segment sounds of notes;
- Construct the performance model, which translates a symbolic score to control signals in amplitude and frequency;
- Construct the instrument model, which synthesizes instrument sound given the control signals.

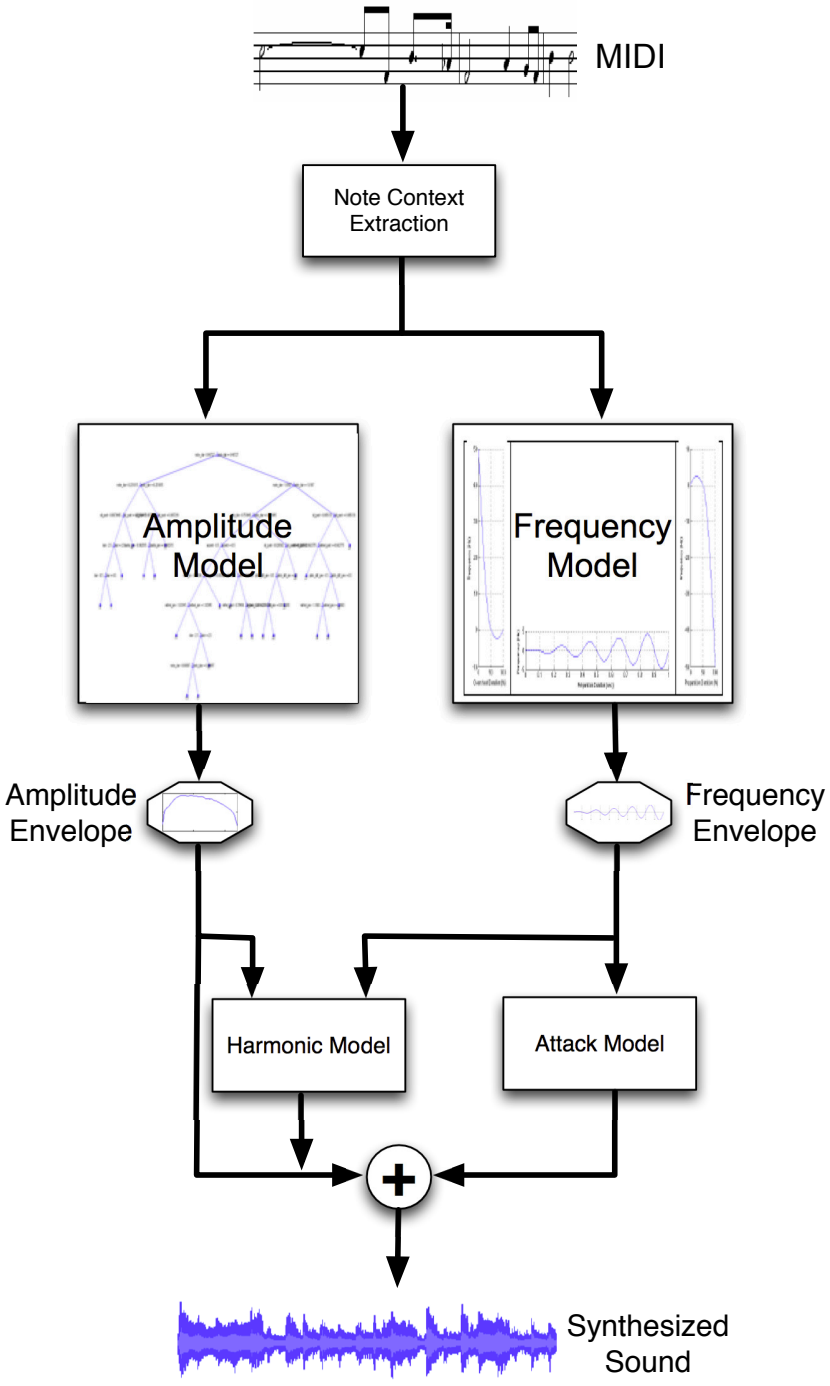


Figure 3.2: Detailed diagram of the synthesis framework

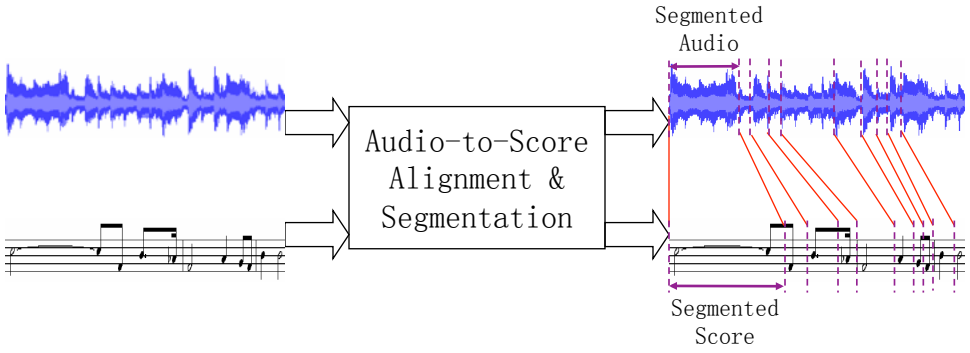


Figure 3.3: Audio-to-score alignment and note segmentation

3.2.1 Generating labeled data

For training purposes, the system needs acoustic examples of individual notes extracted from continuous musical phrases, along with their corresponding symbolic context. But the training inputs of the system are acoustic recordings and their corresponding scores, each representing a whole piece of music. An automated process is needed to extract audio and score context for each note from the entire acoustic and symbolic input sequences. We first align the acoustic recording and the corresponding score in time to find the precise correspondence between them. Then we segment both acoustic recordings and scores into individual notes and output segmented audio clips and score snippets that correspond to each other. As shown in Figure 3.3, this data processing module is an important step. It generates labeled data required by the construction process of the performance model.

The nuances of sound, especially onset attacks, are particularly important for synthesizing realistic instrument sound. As these portions with fine details are usually as short as tens of milliseconds, the precision requirement of the note segmentation is rather high. Though lacking the manually labeled data to start with, we managed to train an accurate audio segmenter with a bootstrapping method. The construction process of such a note segmenter involves audio-to-score alignment and training of a feed-forward neural network as the segmentation model.

3.2.2 Constructing the performance model

The performance model utilizes the context information and generates continuous amplitude and fundamental frequency envelopes accordingly.

The segmented audio generated by the process of audio-to-score alignment and note segmentation is analyzed to acquire the root-mean-square (RMS) amplitude envelope and the fundamental frequency envelope. Curve fitting can be applied on those envelopes to obtain a parameterized representation of the continuous signals. Note that the curve-fitting functions for amplitude and frequency envelopes differ significantly. They are carefully designed to capture the characteristics of the corresponding types of envelopes.

Frequency envelopes are generated using a set of hand-tuned functions, whereas the model for synthesizing amplitude envelopes is automatically learned from training samples. The core of the amplitude model is a decision tree, which finds a representative envelope judging by score context. The model then interpolates the parameters of the representative envelope to re-synthesize an appropriate new envelope, which is adjusted to best fit the note context. The complete process diagram of constructing an amplitude performance model is shown in Figure 3.4.

3.2.3 Building the instrument model

The instrument model accepts control functions of amplitude and frequency as input and produces a digital audio sound as output. The synthesized sound should be perceptually very close to the acoustic instrument being modeled.

The model being built is largely based on the instrument model from Derenyi and Dannenberg's research on Combined Spectral Interpolation Synthesis 2.1.2.2. It is essentially a memory-based approach, and employs simple yet effective techniques, such as table lookup and interpolation. This compact model already works fairly well for the acoustic instruments being modeled, thus our effort here mainly focuses on re-constructing the model and comparing with other ap-

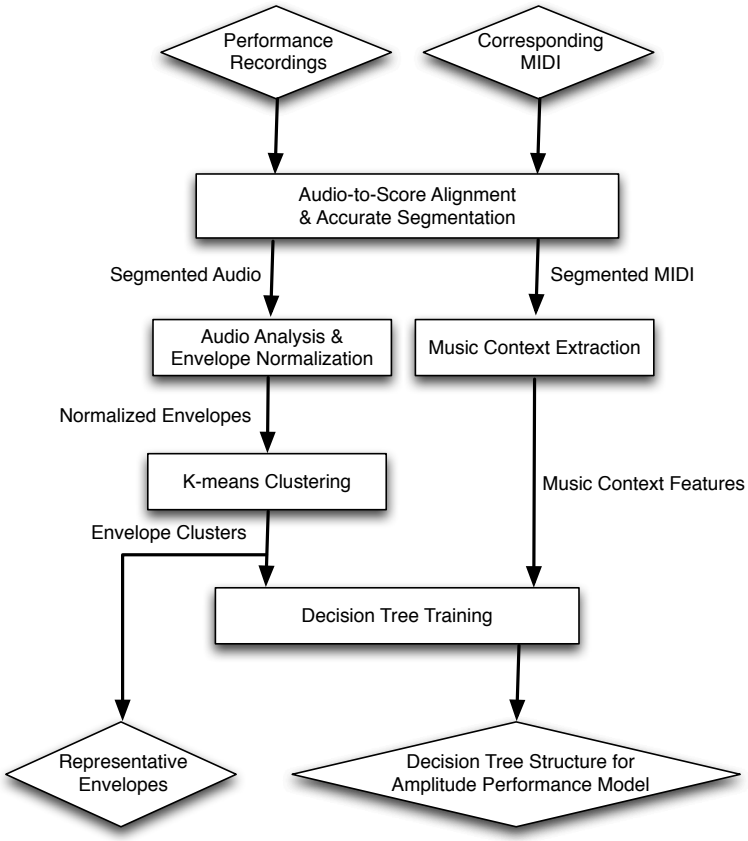


Figure 3.4: Process diagram of constructing an amplitude performance model

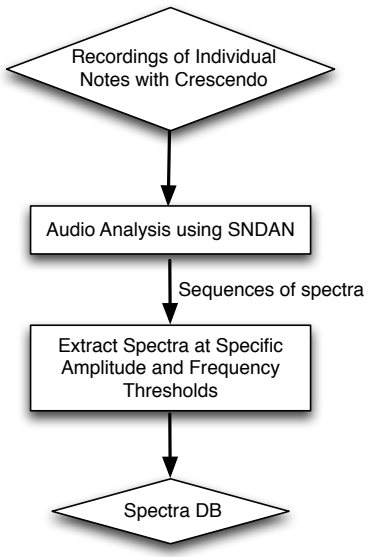


Figure 3.5: Process diagram of constructing the spectra DB for the harmonic model

proaches such as neural network. Also several previously hand-tuned steps, such as choosing and splicing attacks, are automated in the system.

The instrument model consists of two sub-models, one for harmonics and the other for attack sound. The former one synthesizes the harmonic body of the instrument sound, while the latter models the inharmonic attack transients. The attack model contains a set of pre-recorded attacks, while the core of the harmonic model is a database of spectra at specific frequency and amplitude thresholds. The construction process diagram of such a database is shown in Figure 3.5.

In the following chapters, each important module will be described in more detail, i.e. audio-to-score alignment and note segmentation, the instrument model, and the performance model. We will explain the logic behind them, describe their infrastructures, discuss and compare one or multiple approaches for each of them through experiments and live listening tests.

Chapter 4

Audio-To-Score Alignment and Accurate Note Segmentation

In order to learn the properties of amplitude and frequency envelopes for the performance model, we need to segment individual notes from acoustic recordings and link them to corresponding score fragments. This requires an audio-to-score alignment and accurate note segmentation process.

The research of audio-to-score alignment has become a popular Music Information Retrieval (MIR) topic in recent years. Linking signal and symbolic representations of music can enable many interesting applications, such as polyphonic music retrieval [27] [40], real-time score following [50], and intelligent editors [13].

We previously developed a polyphonic audio alignment system and effectively deployed it in several applications [27] [13]. However, we face a particular challenge when trying to use the alignment system in this case, mainly due to the special requirement imposed by the nature of instrumental sounds. For any individual note generated by a musical instrument, the attack part is perceptually very important. Furthermore, attacks are usually very short. The attack part of a typical trumpet tone lasts only about 30 milliseconds (see Figure 4.1). Due to limits imposed by the acoustic features used for alignment, the size of the analysis window is usually

0.1 to 0.25 s, which is too large to accurately pinpoint note onsets. Therefore, we must pursue *accurate* audio alignment with a resolution of several milliseconds. Thus, our work is motivated and characterized by three goals:

(1) very high precision, on the order of milliseconds,

(2) very high accuracy, which can only be obtained through the use of a symbolic score and score alignment, and

(3) no need for manual segmentation to provide training examples.

While previous work addresses some of these requirements, we believe our results are superior for our application.

While pursuring automatic audio alignment with very high precision and accuracy, we discovered that such an approach can be used to train an accurate note segmenter [26], which is a nice added benefit of this thesis research.

Audio Segmentation is one of the major topics in Music Information Retrieval. Many MIR applications and systems are closely related to audio segmentation, especially those that deal with acoustic signals. Audio segmentation is sometimes the essential purpose of those applications, such as dividing acoustic recordings into singing solo and accompaniment parts. Alternatively, audio segmentation can form an important module in a system, for example, detecting note onsets in the sung queries of a Query-by-Humming system.

A common practice is to apply various machine learning techniques to the audio segmentation problem, and good results have been obtained. Some of the representative machine learning models used in this area are the Hidden Markov Model (HMM) [49], Neural Network [38], Support Vector Machine (SVM) [36], and Hierarchical Model [33]. However, as in many other machine learning applications, audio segmentation using machine learning schemes inevitably faces a problem: getting training data is difficult and tedious. Manually segmenting each note in a five-minute piece of music can take several hours of work. Since the quantity and quality of the training data directly affect the performance of the machine learning model, many designers have no choice but to label some training data by hand.

In a sense, audio-to-score alignment and music audio segmentation are closely related. Both operations are performed on acoustic features extracted from the audio, though alignment focuses on global correspondence while segmentation focuses on local changes. Given a *precise* alignment between the symbolic and corresponding acoustic data, desired segments can be easily extracted from audio. Even if alignment is not that precise, it still provides valuable information to music audio segmentation. Conversely, given a *precise* segmentation, alignment becomes almost trivial. This relationship between alignment and segmentation can be exploited to improve music segmentation.

We developed a bootstrap method that uses automatic alignment information to help train the segmenter. The training process consists of two parts. One is an alignment process that finds the time correspondence between the symbolic and acoustic representations of a music piece. The other is an audio segmentation process that extracts note fragments from the acoustic recording. Alignment is accomplished by matching sequences of chromagram features using Dynamic Time Warping (DTW). The segmentation model is a feed-forward neural network, with several features extracted from audio as the inputs, and a real value between 0 and 1 as the output. The alignment results help to train the segmenter iteratively. Our implementation and evaluation show that this training scheme is feasible, and that it can greatly improve the performance of audio segmentation without manually labeling any training data. Though we need to note that the audio segmentation process is aimed at detecting note onsets, this bootstrap learning scheme combined with automatic alignment can also be used for other kinds of audio segmentation.

Because our segmentation system was initially developed for music synthesis, we first set up the experiments with monophonic audio, as that is our main concern. We have extended this work to deal with polyphonic music, just to demonstrate that this approach can also work well in polyphonic cases.

The audio-to-score alignment process is closely related to that of Orio and Schwarz [46], which also uses dynamic time warping to align polyphonic music to scores. While we use the chromagram (described in a later section), they use a measure called Peak Structure Distance,

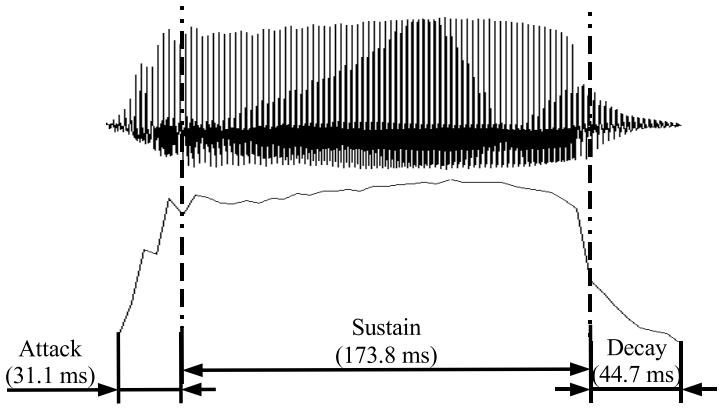


Figure 4.1: A typical trumpet slurred note (a mezzo forte C4 from an ascending scale of slurred quarter notes), displayed in waveform along with the amplitude envelope. Attack, sustain and decay parts are indicated in the figure.

which is derived from the spectrum of audio and from synthetic spectra computed from score data. Another noteworthy aspect of their work is that, since they also intend to use it for music synthesis [55], they obtain accurate alignment using small (5.8 ms) analysis windows, and the average error is about 23 ms [59], which makes it possible to directly generate training data for audio segmentation. However, this also greatly affects the efficiency of the alignment process. They reported that even with optimization measures, their system was running "2 hours for 5 minutes of music, and occupying 400MB memory". In contrast, our system uses larger analysis windows and aligns 5 minutes of music in less than 5 minutes. Although we use larger analysis windows for alignment, we use small analysis windows (and different features) for segmentation, and this allows us to obtain high accuracy.

Note that the best features for score alignment, which typically relies on melody and harmony, may not be the best features for detecting segment boundaries, which are also characterized by rapid changes in energy and noise content. Our approach recognizes this difference and uses different features for segmentation vs. alignment.

In the following sections, we describe our system in detail. We first introduce the audio-to-score alignment process and the segmentation model. Then we describe the bootstrap learning method in detail, and evaluate the system and present some experimental results. We summarize

this chapter and draw some conclusions in the last section.

4.1 Audio-to-score Alignment

4.1.1 The chroma representation

As mentioned above, the alignment is performed on two sequences of features extracted from both the symbolic and audio data. Compared with several other representations, the chroma representation is clearly a winner for this task [27].

Thus, our first step is to convert audio data into *discrete chromagrams*: sequences of chroma vectors. The chroma vector representation is a 12-element vector, where each element represents the spectral energy corresponding to one pitch class (i.e. C, C#, D, D#, etc.). There are many variations on how the chroma representation is computed. In our case, we compute a chroma vector from a magnitude spectrum by assigning each bin of the FFT to the pitch class of the nearest step in the chromatic equal-tempered scale. Only the FFT bins associated with frequencies ranging from 40 Hz to 20 kHz are used. Then, given a pitch class, we average the magnitude of the corresponding bins. This results in a 12-value chroma vector. Each chroma vector in this method represents 50 milliseconds of audio data (non-overlapping). For an acoustic recording with the sampling rate of 44.1 kHz, the analysis window size is 2205 samples, while the FFT size is 4096 as it is the first equal or superior power of 2 of the window size.

The symbolic data, i.e. MIDI file, is also to be converted into chromagrams. The traditional way is to synthesize the MIDI data, and then convert the synthetic audio into chromagrams. However, we have found a simple alternative that directly maps from MIDI events to chroma vectors [27]. To compute the chromagram directly from MIDI data, we first associate each pitch class with an independent unit chroma vector – the chroma vector with only one element value as 1 and the rest as 0. Then for each analysis frame, if there exists a MIDI note, the corresponding chroma vector is computed by multiplying the unit chroma vector associated with the note pitch

with the square of the note velocity; or it is a summation of such chroma vectors if there is polyphony in the frame.

The direct mapping scheme speeds up the system by skipping the synthesis procedure, and it rarely sacrifices any quality in the alignment results. In most cases we have tried, the results are generally better when using this alternative approach. Furthermore, when audio is rendered from symbolic (MIDI) data, there can be small timing variations in practice. Since we are aiming for highly accurate results, it is desirable to bypass this source of error.

4.1.2 Matching MIDI to audio

After obtaining two sequences of chroma vectors from the audio recording and MIDI data, we need to find a time correspondence between the two sequences such that corresponding vectors are similar. Before comparing the chroma vectors, we normalize the vectors, as obviously the amplitude levels vary throughout the acoustic recordings and MIDI files. We experimented with different normalization methods, and normalizing the vectors to have a mean of zero and a variance of one seems to be the best one. But this can cause trouble when dealing with *silence*. Thus, if the average amplitude of an audio frame is lower than a predefined threshold, we define it as a silence frame. We then calculate the Euclidean distance between the vectors. The distance is zero if there is perfect agreement. If either of the two compared chroma vectors is a silence frame, we assign the distance a pre-defined value d_{max}. The value of d_{max} should be carefully selected according to the calculation pattern as shown in Figure 4.3, as it can impact the alignment quality. From our experience, d_{max} should have a value, such as 16, that neither penalizes the horizontal or vertical direction of the path, nor makes matching silence to non-silence frames impossible. Otherwise, the detection of those note onsets immediately after silence frames can be slightly delayed.

Figure 4.2 shows a similarity matrix where the horizontal axis is a time index into the acoustic recording, and the vertical axis is a time index into the MIDI data. The intensity of each point is

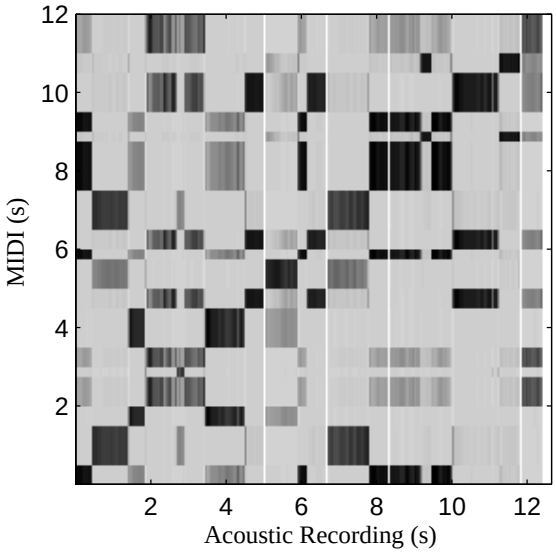
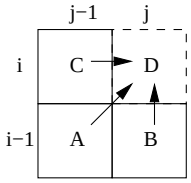


Figure 4.2: Similarity matrix for the first part in the third movement of English Suite composed by R. Bernard Fitzgerald. The acoustic recording is the trumpet performance by Roger B. Dannenberg.

the distance between the corresponding vectors, where black represents a distance of zero.

We use the Dynamic Time Warping (DTW) algorithm to find the optimal alignment. DTW computes a path in a similarity matrix where the rows correspond to one vector sequence and columns correspond to the other. The path is a sequence of adjacent cells, and DTW finds the path with the smallest sum of distances. For DTW, each matrix cell (i,j) represents the sum of distances along the best path from (0,0) to (i,j). We use the calculation pattern shown in Figure 4.3 for each cell. The best path up to location (i,j) in the matrix (labeled "D" in the figure) depends only on the adjacent cells (A, B, and C) and the weighted distance between the vectors corresponding to row i and column j. Note that the horizontal step from C and the vertical step from B allow for the skipping of silence in either sequence. We also weight the distance value in the step from cell A by $\sqrt{2}$ so as not to favor the diagonal direction. This calculation pattern is the one we feel more comfortable with, but the resulting differences from various formulations of DTW [25] are often subtle. The DTW algorithm requires a single pass through the matrix to compute the cost of the best path. Then, a backtracking step is used to identify the actual path.



$$D = M_{i,j} = min(\begin{bmatrix} A \\ B \\ C \end{bmatrix} + \begin{bmatrix} \sqrt{2} \\ 1 \\ 1 \end{bmatrix} \times dist(i,j))$$

Figure 4.3: Calculation pattern for cell (i, j)

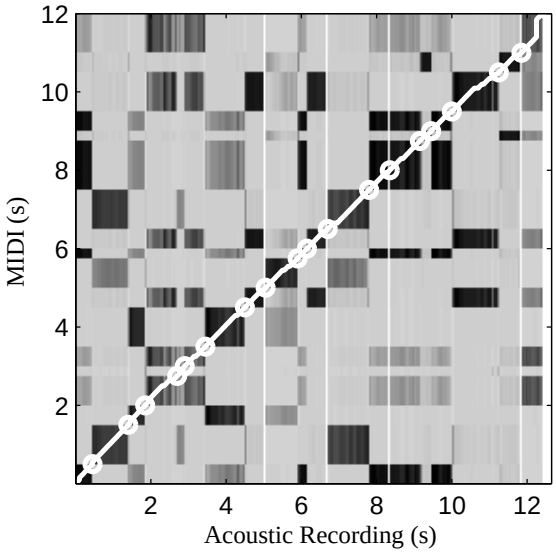


Figure 4.4: The optimal alignment path is shown in white over the similarity matrix of Figure 4.2; the little circles on the path denote the mapping of note onsets.

The time complexity of the automatic alignment is $O(mn)$, where m and n are respectively the lengths of the two compared feature sequences. Assuming the expected optimal alignment path is near the diagonal, we can optimize the process by running DTW on just a part of the similarity matrix, which is basically a diagonal band representing the allowable range of misalignment between the two sequences. Then the time complexity can be reduced to $O(max(m, n))$.

After computing the optimal path found by DTW, we get the time points of those note onsets in the MIDI file and map them to the acoustic recording according to the path (see Figure 4.4).

The analysis window used for alignment is $W_a = 50ms$, and a smaller window actually

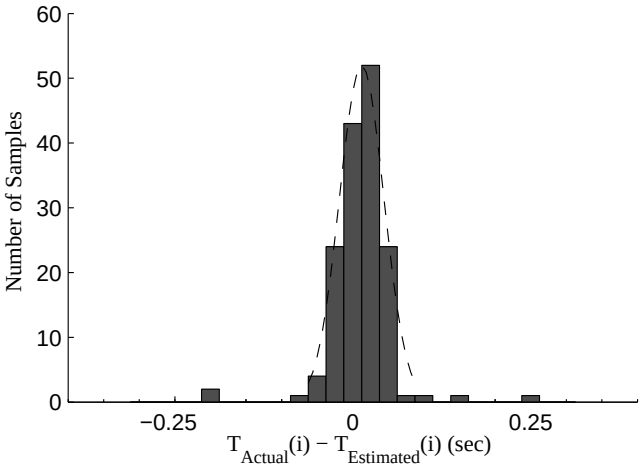


Figure 4.5: Histogram of estimated onset error distribution. $T_{estimated}(i)$ denotes an estimated note onset given by the alignment, while $T_{actual}(i)$ denotes the corresponding actual note onset in acoustic recordings. Here $1 \leq i \leq 154$. The dotted curve represents a Gaussian window with about twice the size of the alignment analysis window ($2 \times 0.05s = 0.1s$).

makes the alignment worse, apparently because large windows give more reliable estimates of the "true" chroma vector, which in turn makes distance estimates more reliable. Thus the alignment result is really not that *accurate*, considering the resolution from alignment is on the same scale as the analysis window size. Nevertheless, the alignment path still indicates roughly where the note onsets should be in the audio. In fact, the error between the actual note onsets and the estimated ones found by the alignment path is similar to a Gaussian distribution. Figure 4.5 shows the histogram of such error distribution from 154 note onset samples. The possibility of observing an actual note onset around an estimated one given by the alignment is approximately a Gaussian distribution. This is valuable information that can help train the segmenter.

4.2 Note Segmentation

4.2.1 Acoustic features

Several features are extracted from the acoustic signals. The basic ones are listed below:

- Logarithmic energy, distinguishing silent frames from the audio,

  $LogEng = 10 \log_{10} \frac{Energy}{Energy_0}$,

  where $Energy_0 = 1$.

- Fundamental frequency $F0$. Fundamental frequency and harmonics are computed using the McAulay-Quatieri Model [39] provided by the

  SNDAN package [3].

- Relative strengths of first three harmonics

  $RelAmp_i = \frac{Amplitude_i}{Amplitude_{overall}}$,

  where i denotes which harmonic.

- Relative frequency deviations of first three harmonics

  $RelDFr_i = \frac{f_i - i \times F0}{f_i}$,

  where f_i is the frequency of the i^{th} harmonic

- Zero-crossing rate (ZCR), serving as an indicator of the noisiness of the signal.

Furthermore, the derivatives of those features are also included, as derivatives are good indicators of fluctuations in the audio such as note attacks or fricatives.

All of those features are computed using a sliding non-overlapping analysis window W_s with a size of 5.8 ms. If the audio to be processed has the sample rate of 44.1 KHz, every analysis window contains 256 samples. Other features that have been proved to be useful in segmentation [6] could be added to this set and might improve performance.

4.2.2 Segmentation model

We use a multi-layer Neural Network as the segmentation model (see Figure 4.6). It is a four-layer feed-forward network, which is fully connected between each layer. Each neuron (perceptron) is a Sigmoid unit, which is defined as $f(s) = \frac{1}{1+e^{-s}}$, where s is the input of the neuron, and $f(s)$ is the output. The input units accept those features extracted from the acoustic

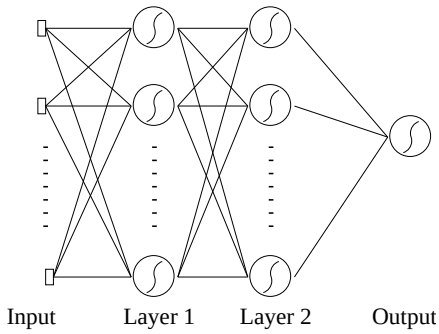


Figure 4.6: Neural network for segmentation

signals. The output is a single real value ranging from 0 to 1, indicating the likelihood of being a segmentation point for the current audio frame. In other words, the output is the model's estimation of the certainty of a note onset. When using the model to segment the audio file, an audio frame is classified as a note onset if the output of the segmenter is more than 0.5.

Neural networks offer a standard approach for supervised learning. Labeled data are required to train a network. Training is accomplished by adjusting weights within the network to minimize the expected output error. We use a conventional back-propagation learning method to train the model.

We should note that the segmentation model used in this approach is a typical but rather simple one, and its performance alone may not be the best among other more complicated models. The emphasis of this approach is to demonstrate that the alignment information can help train the segmenter and improve its performance via bootstrap learning, not how well the standalone segmenter performs.

4.3 Bootstrap Learning

After we get the estimated note onsets from the alignment path found by DTW, we create a probability function indicating the possibility of being an actual note onset at each time point in the acoustic recording. As shown in Figure 4.7, the function is generated by summing up a set of Gaussian windows. Each window is centered at the estimated note onsets given by the alignment

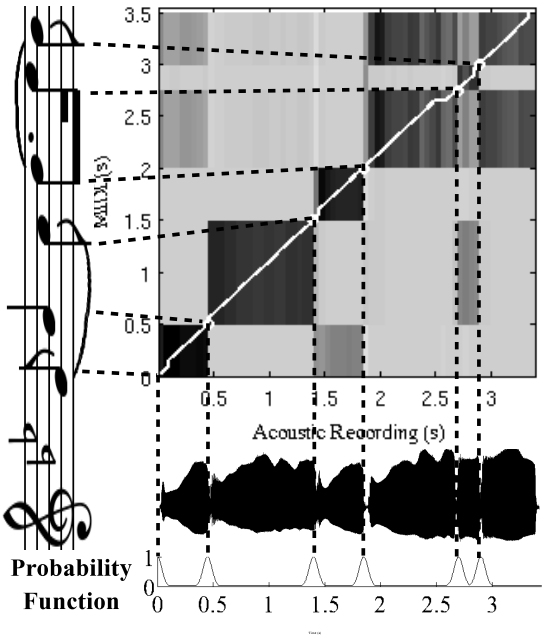


Figure 4.7: Probability function generated from the alignment of a snippet, which is a phrase of the music content in Figure 4.2.

path, and has twice the size of the alignment analysis window ($2 \times 0.05s = 0.1s$). For those points outside any Gaussian window, the value is assigned to a small value slightly bigger than 0 (e.g. 0.04).

The creation of the probability function is based on the assumption that note onsets are independent of each other. We should also point out that this is just a pseudo-probability function. The mathematical definition of probability function $p(x)$ requires that the sum of $p(x)$ of all possible values of x is 1, while the sum of our probability function is close to but bigger than the number of note onsets in the acoustic recording. Of course, the function can be normalized to have the sum of 1. But given the way it is used here, it does not matter whether the function is normalized or not.

Then we run the following steps iteratively until the outputs of the trained neural network are unchanged by a round of training.

1. Execute segmentation process on the acoustic audio.

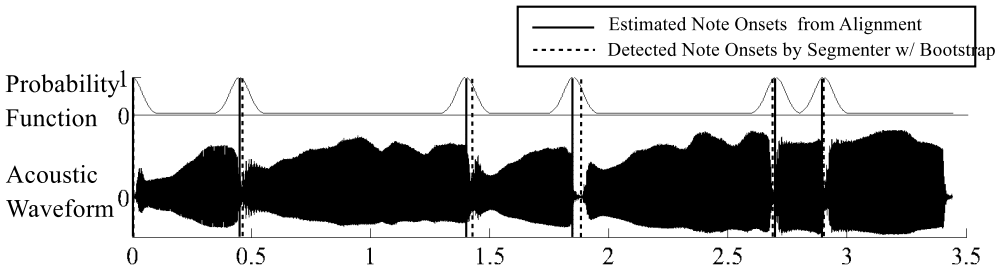


Figure 4.8: Note segmentation results on the same music content as in Figure 4.7. Note that the note onsets detected by the segmenter with bootstrapping are not exactly the same as the ones estimated from alignment. This is best illustrated on the note boundary around 1.8 seconds.

2. Multiply the sequence of real values v output by the segmenter with the note onset probability function. The result is a new sequence of values denoted as v_{new}.

3. For each estimated note onset, find a time point that has the biggest value v_{new} within a window W_p, and mark it as the adjusted note onset. The window is defined as follows:
$W_p(i) = [max\left(\frac{T_i+T_{i-1}}{2}, T_i - W_a\right),$
$min\left(\frac{T_i+T_{i+1}}{2}, T_i + W_a\right)],$
where T_i is the estimated onset time of the i^{th} note in the acoustic recording given by alignment, and W_a is the size of the analysis window for alignment.

4. Use the audio frames to re-train the neural network. The adjusted note onset points are labeled as 1, and the rest are labeled as 0. Because the number of positive examples is far less than the negative ones, we adjust the cost function to increase the penalty when false negatives occur. We also use 5-fold cross-validation to prevent data over-fitting.

As the segmentation model has a finer resolution than the alignment model, the trained segmenter can detect note boundaries in audio signals more precisely, as demonstrated in Figure 4.8.

4.4 Evaluations

4.4.1 Monophonic experiments

As mentioned in previous sections, the system was initially designed for helping with music synthesis research. Thus we were mainly focusing on monophony. The experimental data for monophonic music is the English Suite composed by R. Bernard Fitzgerald [18] for Bb Trumpet. This is a set of 5 English folk tunes artfully arranged into one work. Each of the 5 movements (recorded without accompaniment) is essentially a monophonic melody, and the whole suite contains a total of 673 notes. We have several formats of this particular music piece, including MIDI files created using a digital piano, the real acoustic recordings performed by the advisor of this thesis, and synthetic audio generated from the MIDI files.

We made some experiments to compare two systems. One is a baseline segmenter, which is pre-trained using a different MIDI file and its synthetic data; the other is a segmenter with the bootstrap method. Its initial setup is essentially the same with the baseline segmenter. Then the alignment information is used to help iteratively train the segmenter with the bootstrap method. We run the baseline segmenter through all the audio files in the data set and compare its detected note onsets with the actual ones. For the segmenter with bootstrapping, we use cross-evaluation. In every evaluation pass, 4 MIDI files and the corresponding audio files are used to train the segmenter through the bootstrap approach, and the remaining MIDI-audio files pair is used as the test set. This process is repeated so that the data of all 5 movements have been used once for evaluation, and the results on the test sets are combined to evaluate the model performance. It is important to note that *when testing the segmenter with bootstrap learning, score alignment is not used.* If the score were used (and sometimes this is possible), the miss rate and spurious rate would normally be zero because the one-to-one alignment of audio notes to symbolic notes leaves little opportunity for error.

We calculate several values to measure the performance of the systems. Miss rate is defined

Table 4.1: Model Comparison on Synthetic Monophonic Audio

Model	**Miss Rate**	**Spurious Rate**	**Average Error**	**STD**
Baseline Segmenter	8.8%	10.3%	21 ms	29 ms
Segmenter w/ Bootstrap	0.0%	0.3%	10 ms	14 ms

Table 4.2: Model Comparison on Real Monophonic Recordings

Model	**Miss Rate**	**Spurious Rate**	**Average Error**	**STD**
Baseline Segmenter	15.0%	25.0%	35 ms	48 ms
Segmenter w/ Bootstrap	2.0%	4.0%	8 ms	12 ms

as the ratio of missed ones among all the actual note onsets – an actual note onset is determined to be a missed one when there is no detected onset within the window W_p around it; spurious rate is the ratio between spurious onsets detected by the system and all the actual note onsets – spurious note onsets include those detected that do not correspond to any actual onset. Average error and standard deviation (STD) indicate the attribute of the distance between each actual note onset and its corresponding detected onset, if the note onset is neither missed nor spurious.

We first use the synthetic audio from MIDI files as the data set, and the experimental results are shown in Table 4.1.

We also tried the two segmenters on the acoustic recordings. However, it is very difficult to take overall measures, as labeling all the note onsets in acoustic recordings is too time consuming. Therefore, we randomly picked a set of 100 note onsets throughout the music piece (20 in each movement), and measured their results manually. The results are shown in Table 4.2.

The baseline segmenter performs worse on the real recordings than on the synthetic data, which indicates there are indeed some differences between synthetic audio and real recordings

Table 4.3: Model Comparison on Synthetic Polyphonic Audio

Model	**Miss Rate**	**Spurious Rate**	**Average Error**	**STD**
Baseline Segmenter	10.8%	12.1%	28 ms	61 ms
Segmenter w/ Bootstrap	1.2%	0.4%	12 ms	20 ms

that can affect the performance. Nevertheless, the segmenter with bootstrapping continues to perform very well on recordings of an acoustic instrument.

4.4.2 Polyphonic experiments

We also tried the bootstrap method on polyphonic music. The experimental data are 3 piano pieces. They are F. Chopin, Nocturne Op. 9, No. 2; L. van Beethoven, Adagio "Moonlight" Sonata; and W.A. Mozart, Turkish March. As in the monophonic experiments, we used several formats of the pieces, including MIDI files, synthetic audio generated from MIDI files, and acoustic recordings that were performed by myself.

The polyphonic experiments are very similar to the monophonic experiments, and we count multiple simultaneous note onsets as one. Table 4.3 and 4.4 show the experimental results on synthetic audio and acousitic recordings respectively. Compared to experimental results on monophonic music, the polyphonic results are slightly worse with both segmenters, in spite of the fact that piano onsets are generally considered more prominent and easier to detect than trumpet onsets. Maybe it is because with many sounds interfering with each other in the signal, the segmenter sometimes gets confused. Nevertheless, the segmenter with bootstrapping significantly outperforms the baseline segmenter, and its results are comparable to other published work [38]. This indicates that our bootstrap method works well with polyphony.

Table 4.4: Model Comparison on Real Polyphonic Recordings

Model	**Miss Rate**	**Spurious Rate**	**Average Error**	**STD**
Baseline Segmenter	16.0%	17.0%	27 ms	55 ms
Segmenter w/ Bootstrap	5.0%	3.0%	11 ms	23 ms

4.5 Summary

Music segmentation is an important step in many music processing tasks, including beat tracking, tempo analysis, music transcription, and music alignment. However, segmenting music at note boundaries is rather difficult. In real recordings, the end of one note often overlaps the beginning of the next due to resonance in acoustic instruments and reverberation in the performance space. Even humans have difficulty deciding exactly where note transitions occur. One promising approach to good segmentation is machine learning. With good training data, supervised learning systems frequently outperform those created in an ad hoc fashion. Unfortunately, we do not have very good training data for music segmentation, and labeling acoustic recordings by hand is very tedious and time consuming.

Our work offers a solution to the problem of obtaining good training data. We use music alignment to tell us *(*approximately) where to find note boundaries. This information is used to improve the segmentation, and the segmentation can then be used as labeled training data to improve the segmenter. This bootstrapping process is iterated until it converges.

Experiments show that segmentation can be dramatically improved using this approach. Note that while we use alignment to help train the segmenter, we tested the trained segmenters without using alignment. Of course, whenever a symbolic score is available, even more accurate segmentation should be possible by combining the segmenter with the alignment results.

In summary, we have described an approach for music segmentation that uses alignment to provide an initial set of labeled training data. A bootstrap method is used to improve both

the labels and the segmenter. Segmenters trained in this manner show improved performance over a baseline segmenter that has little training. Our bootstrap approach can be generalized to incorporate additional signal features and other supervised learning algorithms.

This method is used to segment acoustic recordings for the very purpose of this thesis research. We believe many other applications can also benefit from this new approach.

Chapter 5

Instrument Model

The instrument model accepts a set of control signals as input and produces a sequence of digital audio sound as output. The goal is to generate realistic instrument tones given the proper control signals. In order to insure the success of the whole synthesis model, the sound produced should be perceptually as close to the acoustic instrument being modeled as possible.

The instrument model is an extended version of the early Spectrum Interpolation Synthesis [57]. It can also be viewed as a specialization of additive synthesis [48] or a generalization of wavetable synthesis [42]. It is based on the observation that the sounds of many musical instruments can be described by the combination of harmonic and additive noise signals [58]. Thus we construct the instrument model with two sub-parts, the harmonic model and the attack model.

For trumpet, the harmonic model accounts for most of the instrument sounds, while the inharmonic model is limited to only tens of milliseconds at the beginning of some notes. Still, the inharmonic portion has significant perceptual effects. Without the attacks generated by trumpet mechanics and air flow, the synthesized trumpet sound just will not sound realistic. The harmonic model is in charge of synthesizing the harmonic tone, while the attack model takes care of the attack transients. The final audio output is a spliced result of the two.

5.1 Harmonic Model

Many successful attempts in music synthesis only focus on the harmonics and ignore the noise part. A classic example is additive synthesis [43]. It models sounds as summations of deterministic sinusoidal components (the partials). A common constraint imposed here is that frequencies of the partials are all multiples of a fundamental frequency. We only focus on modeling traditional acoustic instruments in this thesis work, whereas tone harmonicity is one of its key characteristics. Our approach is therefore not appropriate for certain sounds such as bells, metal bars, and most other percussion sounds. Thus it is generally safe to assume that at every time-point the spectrum is nearly harmonic, with an exception at the attack portion of a note. The instantaneous harmonic spectrum can be expressed by a small number of parameters, called modulation sources. The primary modulation sources are the current RMS amplitude and fundamental frequency. They can be automatically extracted from real performances as well as generated from the digital scores by the performance model. The former can drive the instrument model and synthesize the sound, which can be directly compared with the actual performance.

As described in the related work chapter, additive synthesis is compact in terms of data storage but computationally expensive; whereas the computation process of concatenative synthesis is simple and straightforward, but its sample corpus takes up a lot of memory/storage. Here the harmonic model employs the wavetable interpolation technique, as it is well suited for synthesizing quasi-periodic musical tones, and it can be as compact in data storage requirements and as general as additive synthesis but requires much less real-time computation.

The block diagram of the harmonic model itself can be partitioned into three sequential steps illustrated by the dotted lines shown in Figure 5.1.

The first step takes the control signals as input, and outputs spectral characteristics, notably several harmonic partials and their amplitudes (the spectra). Given the spectrum information from the first step, the second step generates a wavetable [42], which represents one period of the periodic sound in the time domain; the third step produces every sound sample by interpolation

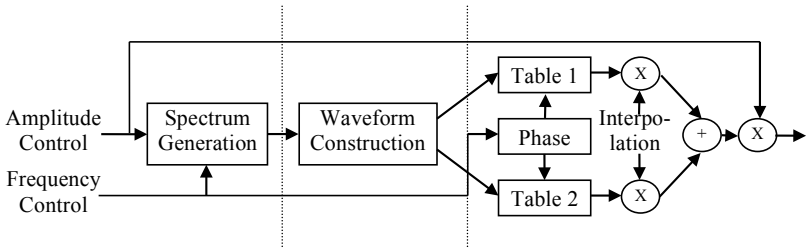


Figure 5.1: Block diagram for the harmonic model.

between two adjacent wavetables. Similar spectral interpolation techniques have been proven successful for synthesizing realistic trumpet tones [15].

5.1.1 From control signals to spectra

The spectra are computed according to the following steps:

1. Record a set of sounds, where each embodies a simple note in a distinct pitch and the amplitude level is either increasing or decreasing at a moderately fast speed, covering as wide a dynamic range as possible. The pitches are pre-defined.

   For Bb trumpet, 19 note samples with steady pitch ranging from $A\flat 3$ (in trumpet scale, corresponds to MIDI pitch 54) to $B5$ (MIDI pitch 81) were recorded, and each has increasing amplitude level from pp to ff. Actually, we tried recording notes with the amplitude level either increasing or decreasing, and found that the former is easier to control the dynamics and results in more ideal recorded samples. Nonetheless previous study [15] shows that the spectrum is independent of whether the RMS amplitude is increasing or decreasing.

2. Obtain a spectrogram for each sound, where each spectrum frame corresponds to a period.

   The spectrogram of each sound was measured using the SNDAN utility package, implemented by Beauchamp et al. [3]. We used SNDAN's phase vocoder method to extract spectra, RMS amplitude, and fundamental frequency. The phase vocoder method employs pitch-synchronous Fourier Transforms over two windowed periods of the waveform to measure the instantaneous spectrum. It is best suited for analyzing sound with fairly

steady pitch and timbre. The output contains a series of analysis frames defining the amplitude and frequency of a harmonic, relative to a fundamental frequency of a given pitch. Each frame is a snapshot of the state of the sound at a very small instant in time.

3. Scan through the spectrogram and retain only spectra at which the amplitude function crosses predetermined thresholds.

   Since each note has increasing amplitude level from quietest to loudest, we can slice the corresponding spectrogram at specific RMS amplitude levels to obtain the spectra. We apply the same measurement on several recorded notes. Now we have a database of spectra, each at a different amplitude and fundamental frequency. When we examine those slices of spectra, we can see that notes with lower pitches have more harmonics in the extracted spectrum, while the ones with higher pitches have fewer. To simplify the implementation and also to reduce computational and storage consumption, we keep 25 harmonics for each spectrum. Listening tests show that the quality of the synthesized sound is not sacrificed when comparing synthesized phrases using 25 harmonics or the original number of harmonics extracted from SNDAN.

   In addition, since phase information is not used in the harmonic model, we only store the relative amplitudes of the harmonics in the spectrum, ignoring the phase information.

Those spectra are then indexed into a two-dimensional lookup table by their corresponding pitches and RMS amplitudes. When accessing this two-dimensional lookup table by the instantaneous amplitude along one dimension and the pitch along the other, the system interpolates among four nearest spectrum samples to yield an output spectrum.

Wessel, Drame, and Wright also studied and compared two other synthesis techniques [66]. One is the memory-based approach. It stores in the memory a set of the spectral information as data points in an n dimensional space; the dimension indexes are their corresponding frequency, RMS amplitude, and etc. The input is a vector indicating a specific point in that space. To generate the output corresponding to each input value, it chooses the k nearest neighboring data

points, weights each of them as a function of the distance between input point and itself, and computes the output by averaging the weighted k selected data points. It is a very flexible model and relatively easy to modify for different requirements. We should point out that the spectral interpolation technique introduced before can be deemed as a special case of the memory-based approach, as here the dimension n is 2 (fundamental frequency and RMS amplitude), k is 4, and the weighting function of the distance between the input and the data points is linear interpolation. The other method described in their paper utilizes a feed-forward neural network with multiple layers. The input units accept the frequency and amplitude functions, and the output units produce the frequencies and amplitudes of the sinusoidal components. The model can be trained with a back-propagation learning method. Unlike the memory-based approach, it does not need to make assumptions about either the distance function or the weighting function, which is its advantage. Thus, the neural network model is very compact and appears to generalize well.

The authors reported that both the neural-network and memory based models functioned well in a real-time context, and in general the neural network model provided a smoother sounding result than the memory-based model. We also implemented the neural network model and compared it to the spectral interpolation model. Both work generally well, while the difference is not quite audible. Since the instrument model is not the primary focus this thesis work, We did not do much tuning on the neural network model given the time constraint.

5.1.2 From spectrum to wavetables

A wavetable represents one period of the periodic sound in the time domain. There are several ways to compute wavetables from the spectra, such as IDFT, IFFT, etc. Here we use the simplest harmonic additive synthesis, which is a simple summation of sinusoids, each representing a harmonic partial and weighted by the corresponding relative amplitudes. It can be mathematically expressed as:

$$W_n[m] = \sum_{k=0}^{K-1} a_{n,k} \cos(\frac{2\pi}{K} km + \theta_{n,k})$$

where $W_n[m]$ is the n^{th} synthesized wavetable with the length M, $a_{n,k}$ and $\theta_{n,k}$ are the amplitude and the phase offset of the k^{th} harmonic partial of a total of K harmonics in the wavetable.

5.1.3 From wavetables to sound samples

As the computing process from control signals to wavetables is relative expensive, it is technically applicable but inefficient to compute a wavetable for every period of the synthesized sound. Here we compute only 20 wavetables per second and interpolate between two tables to generate every sample in between. This produces a smooth and continuous spectral change. The interpolation process between adjacent wavetables $W_{n-1}[m]$ and $W_n[m]$ can be mathematically expressed as:

$$W[m](t) = x(t) \sum_{k=0}^{K-1} a_{n-1,k} \cos(\frac{2\pi}{K} km + \theta_{n-1,k}) + (1 - x(t)) \sum_{k=0}^{K-1} a_{n,k} \cos(\frac{2\pi}{K} km + \theta_{n,k})$$

where $x(t)$ is the interpolation control factor, which is the relative position of time frame t within the time distance between two adjacent wavetables.

When interpolating between two wavetables whose corresponding harmonics are out of phase, phase cancellation and phase shifting cause unintentional but perceptible timbre change, often perceived as a frequency shift. In order to avoid such undesirable results during the interpolation, wavetables are created with matching phases, which means the corresponding harmonics in adjacent wavetables $W_{n-1}[m]$ and $W_n[m]$ are in phase, i.e. $\theta_{n-1,k} = \theta_{n,k} = \theta_k$. Thus, we have

$$W[m](t) = \sum_{k=0}^{K-1} (x(t)a_{n-1,k} + (1 - x(t))a_{n,k}) \cos(\frac{2\pi}{K} km + \theta_k)$$

Hence the effective amplitude of the k^{th} harmonic in the interpolated wavetable $W(t)$ is

$$a_k(t) = x(t)a_{n-1,k} + (1 - x(t))a_{n,k}$$

So when constraining the corresponding harmonics of each generator wavetable to be in phase, the amplitude of a harmonic of the output at sample t equals the linear interpolation of the amplitude of the respective harmonic in the adjacent wavetables.

The phase information plays an crucial role when combined with the attack model, but here in the harmonic model it does not have a significant audible effect on the synthesized harmonic sound. Thus the phases are simply determined by the spliced attacks, as described in the attack model section.

To capture rapid amplitude changes in the sound, especially between slurred notes, the synthesized output from wavetable interpolation is then scaled by the instantaneous RMS amplitude to produce the proper amplitude fluctuations in the sound. The amplitude signal is realized as a piecewise linear curve, with 100 breakpoints per second.

The synthesized sound from the harmonic model now has controlled fluctuations in pitch, amplitude and timbre.

5.2 Attack Model

The residual component refers to the stochastic non-harmonic part of the sound. It is an important factor that should not be simply overlooked, though that hugely depends on the characteristics of individual instruments. Some noisy instruments, such as a flute, may require careful consideration in designing and incorporating a residual model. But for the trumpet, the inharmonicity in general can be pretty much ignored except for the attack portion at the beginning of each note, which represents the transitional part from silence or complete inharmonicity to harmonic partials. Previous study [15] demonstrated that attacks are particularly important for

synthesizing convincing trumpet tones. Its inharmonic attack portion in notes is one of the identifying characteristics of the instrument. Thus we do not need to pay much attention to the general inharmonicity throughout each note, and instead focus more on the attack model.

We chose to use recorded attacks to give the impression of a natural attack. For trumpet, attacks begin with a stopped airflow and silence, which clearly marks the beginning of the attack portion. Thus we only need to splice from the attack to the tone. The splicing process is as follows:

1. Extract recorded attacks from the training data. We first find note onsets from the results of audio-to-score alignment and accurate onset segmentation. This is accomplished by data pre-processing as described in earlier chapter. For those note onsets that start with silence, cut an audio slice from the onset. The duration of the slice should be generally short. Shorter attacks are preferred because they require less memory consumption, and they are easier to be attached to different envelope shapes as they themselves do not contain much envelope shape information. On the other hand, the attack duration should also be long enough that the attack starts with noisy and inharmonic beginning but ends with harmonic structure, and the overall RMS amplitude, amplitude and phase of each partial at the end of the attack can be measured by the analysis software. Experiments show that extracted attacks with 30ms duration is good enough for realistic trumpet synthesis. Such sampled attacks should be extracted for each step in the synthesis pitch range.

   Experience also shows that there is not a wide range of attacks in terms of dynamics, rate, envelope, or brightness. Thus, at least for conventional, classical trumpet playing, relatively few attack samples are needed.

2. When synthesizing a trumpet note, find the sampled attack corresponding to the note pitch, and use the phase distribution measured at the end of the attack to compute all the wavetables in the subsequent spectrally interpolated sound until silence or the next attack. This ensures a smooth transition as phases match at the splice point, i.e. the end of the recorded

attack and the beginning of the synthesized harmonic tone. This way, phase cancellation is automatically avoided.

Experience has shown that this phase matching is crucial, so for example, a simple cross-fade from attack to harmonic model without matching phases is not acceptable.

3. Use the amplitude distribution measured at the end of the attack to generate the first wavetable in the harmonic model. The rest are still generated using the normal spectral interpolation technique. This avoids any audible clicks or amplitude discontinuity at the splice points.

4. Linearly scale the amplitude of the entire attack so that the amplitude at the end of the attack is equal to the amplitude from the performance model at the splice point.

Attacks do not need to be attached to every synthesized note. In real trumpet performance, usually attacks are audible when the notes being played are tongued onsets, which means they should all begin with a stoppage of airflow and a definite silence. So we only introduce attacks at the beginnings of phrases. And this initial attack further determines all the harmonic phases within the phrase. This makes the splicing process easier, as we never need to splice to the beginning of an attack due to the silence. In the case of slurs and legato transitions, attacks are simply omitted.

5.3 Evaluation

To test the instrument model, we use the original RMS amplitude and fundamental frequency control signals (modulation sources) measured from real performances, and run through the instrument model to render the synthesized version. Listening tests show that, given the proper modulation sources, we can synthesize realistic trumpet performances.

In fact, the ability to drive the synthesis model using parameters extracted from acoustic performances is one of the strengths of this approach. Without this property, we could not evaluate

and refine the model. For example, if we were to drive the model with traditional ADSR envelopes, the results would be poor. This may explain why physical models have not gained more traction: regardless of how good the model, synthesis cannot produce good results without good control, and physical models do not offer any direct methods to obtain good control parameters.

Chapter 6

Performance Model

If the instrument model provides us with a means to synthesize realistic sound from modulation sources, we now need a means to determine modulation sources. The goal of the performance model is to automatically generate proper control signals of amplitude and fundamental frequency (modulation sources) for the instrument model, based solely on features extracted from a symbolic score.

The musical score is represented in MIDI for the system to understand and process. Music context information is extracted from MIDI and used as the input of the performance model. Alternatively, we can extract score information from MusicXML [20], a more complete and elaborate representation of the musical score.

Most sampling-based synthesizers only get some coarse note features of pitch, duration, and loudness as input. But that is not enough for our purpose of creating natural-sounding wind instrument synthesis. The performance model here focuses on the fine details of control signals, including appropriate amplitude and frequency envelopes, slurs, vibrato, attacks, and other audible effects. Those fine details are obtained by analyzing information residing in music structure, the relations inside or between the phrases, and the interactions among notes. The rendered audio is a straightforward or "stick-to-the-book" type of interpretation, not an expressive or creative performance involving personal style and variation or improvisation. We leave the exploration

of expressive styles and emotion to future work.

Even though many musical instruments such as a trumpet are complicated dynamic systems, it appears that the system behavior is almost entirely characterized by the amplitude and fundamental frequency at any moment. There are dramatic and systematic changes to amplitude and frequency envelopes depending upon how the note is articulated, whether the note is part of an ascending or descending line, and other factors relating to the context of the note. A quarter note followed by a rest is played differently from one followed by another note. Previous experiments by Dannenberg et al. [11] also verified that, by properly modeling amplitude envelopes, very convincing trumpet tones can be synthesized.

Dannenberg and Derenyi's study on trumpet envelopes [11] successfully designed functions mapping from features in the score to amplitude envelopes. Though the mappings/functions tuning were done by hand, the way they approached the problem shows a promising future for using machine learning techniques to train on performance examples and learn the mappings automatically. This lays the foundation of this chapter.

In the following sections we will first describe how to extract music context from score, and then discuss the models that learn and construct amplitude and frequency envelopes. We will also briefly introduce some of our previous attempts over the years to tackle this issue, and compare them with the current approach.

6.1 Music Context

As mentioned earlier, here the musical score is represented in MIDI (Musical Instrument Digital Interface), which is a widely adopted technical standard that describes, controls and connects electronic music instruments, computers, and other related devices [62]. It consists of a sequence of event messages, which specify pitch and velocity, control signals for parameters such as volume, vibrato, audio panning and cues, and clock signals that set and synchronize tempo between multiple devices [28]. However, in practice many rich control signals are missing when MIDI is

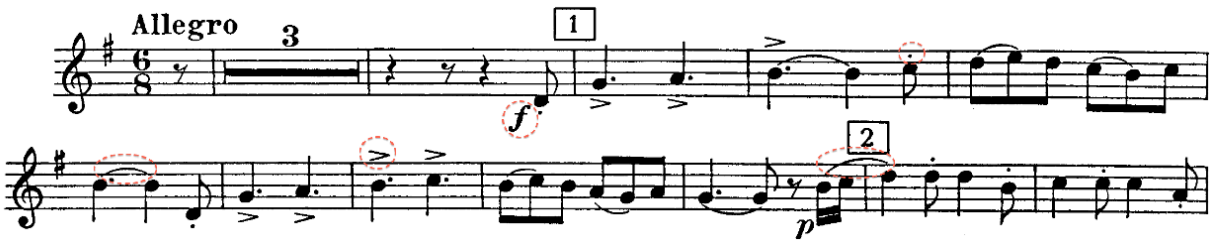


Figure 6.1: A fragment from the scanned score of the English Suite Movement V: Finale. Typical annotations to be extracted are circled in red.

used for representing digital scores, and only the basic MIDI events remain: pitch, velocity, start and end time of each note. Still the context of notes within phrases, and structural relationship among adjacent notes can be encoded to some extent within the simple note information stored in MIDI. For example, we need to know what notes are slurred (no tongued attack) and which are tongued. We use the simple and common encoding that tongued notes have at least a slight gap between the end of the previous note and the beginning of the note in question, whereas slurred notes have no gap or even a slight overlap in the MIDI data.

Musical annotations are important for rendering realistic audio recordings. For example, slurs and staccatos affect the actual duration of notes in performance. Rather than manually encoding annotations into MIDI data, we can obtain them from a MusicXML representation of the score. MusicXML [20] is an XML-based file format for representing western musical notation. It is a more complete and elaborate representation of digital sheet music. We used the software SharpEye [65] to process the scanned copy of the score and generate corresponding MusicXML data. It consists of simple annotations such as music dynamics, note articulations including fermata, accent, staccato, and tenuto, and relationship among notes such as ties and slurs, plus crescendo and diminuendo wedges, as illustrated in Figure 6.1.

For a note n_i in a typical MIDI file, we have simple information such as start time $start_i$, end time end_i, pitch key_i, and velocity vel_i. Though the information is rather simple and limited, we can still extract music context information by comparing adjacent notes/rests. A set of parameters is generated as following to represent the context of a note:

- Note duration $dur_i = end_i - start_i$;
- Note pitch in MIDI pitch value key_i;
- Note IOI, i.e. interval between the start time of the next note and the current one: IOI_i;
- Leading silence in seconds: $start_i - end_{i-1}$. The value 1 is used for the first note in the score;
- Following silence in seconds: $start_{i+1} - end_i$. The value 1 is used for the first note in the score;
- Duration and IOI ratio of the note itself and among adjacent notes:
  - $\frac{dur_i}{IOI_i}$;
  - $\frac{dur_i}{IOI_{i-1}}$, 0 if the note is the first one in the score or has long leading silence ($\geq$ 1 second);
  - $\frac{dur_i}{IOI_{i+1}}$, 0 if the note is the last one in the score or has long following silence ($\geq$ 1 second);
- Pitch differences among adjacent notes, in MIDI pitch value:
  - Compared with previous note: $key_i - key_{i-1}$, 0 if the note is the first one in the score or has long leading silence;
  - Compared with next note: $key_{i+1} - key_i$, 0 if the note is the last one in the score or has long following silence;
- Note velocity vel_i;
- Velocity ratios among adjacent notes:
  - Compared with previous note: $\frac{vel_i}{vel_{i-1}}$;
  - Compared with the next note: $\frac{vel}{vel_{i+1}}$.

Note velocity vel_i determines the maximum amplitude of the note. But sometimes such information is not available in MIDI, i.e. all notes in the MIDI have the same velocity value. When

doing side-by-side comparison between the original and synthetic audio, we use the maximum amplitude of notes extracted from the original acoustic recordings as the velocity for simplicity. For synthesizing music purely based on the score, we use the annotation information extracted from the MusicXML data, and apply a set of manually defined rules to interpret those annotations into note velocity.

Some typical rules to encode score information into MIDI include:

1. Accent: $vel_i = vel_i \times 1.2$;
2. Dynamics ($pp, p, mp, mf, f, ff \ldots$): all the notes following a dynamic indication are assigned with a corresponding default velocity value.
3. Crescendo and diminuendo wedges: An increasing/decreasing multiplying factor is applied one by one to the velocity of all notes in the range.
4. Staccato: $dur_i = \max(0.1, dur_i \times 0.6)$;
5. Tenuto: $dur_i = dur_i - silence_{min}$;
6. Fermata: $dur_i = dur_i \times 2$;
7. Slur: for all the notes in neither the beginning nor middle of a slur, $dur_i = dur_i - silence_{norm}$.

Now we have the right representation of digital scores as the input. If the MusicXML data is available, some annotation information such as accent, staccato, fermata, and slur type (beginning, middle or end) can be directly included in the feature set of the note context. They proved to help achieve better classification results mapping from music context to representative envelopes, as described later in the chapter.

6.2 Amplitude Envelope

Sounds made by musical instruments vary in volume over the entire duration of the sound, and synthesized sounds do the same. The amplitude of a typical sound thus can be segmented

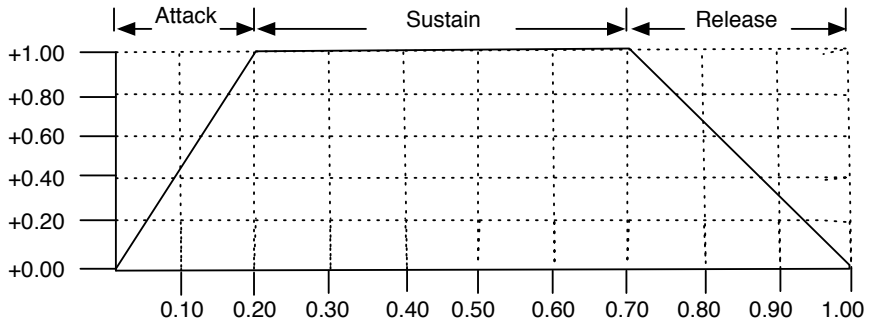


Figure 6.2: A simple envelope with attack, sustain and release.

into several portions. Attack refers to the initial rise in volume, then followed by a steady state portion called sustain, and a final segment that decays to silence, named release. A typical envelope is shown in the following graph (Figure 6.2).

Instantaneous amplitude is of course the audio waveform, so a common way to measure volume variation is to compute the RMS (root mean square) amplitude over the course of each fundamental period of vibration. This gives a constant amplitude that would transfer the same amount of energy as the fine varying audio waveform. RMS amplitude r is calculated as below:

$$r = \sqrt{\frac{1}{N}\sum_{i=1}^{N} a_i}$$

Here N denotes the number of samples in an audio signal to be computed, and a_i is the the amplitude of the i^{th} sample. For a sine wave, the RMS amplitude is $\frac{1}{\sqrt{2}}$ $(\simeq 0.707)$ times the peak amplitude value. By interpolating RMS values, we can obtain an envelope of the signal.

A basic envelope consists of only attack, sustain and release segments, but we often need to produce more complex envelopes. A four segment ADSR type envelope [1] as shown in Figure 6.3 is commonly used in traditional synthesizers. Still, these simple envelopes are way too coarse for natural-sounding music synthesis, so a more sophisticated model with more parameters is required.

Clynes suggested that [9] amplitude envelopes should be modified according to context.

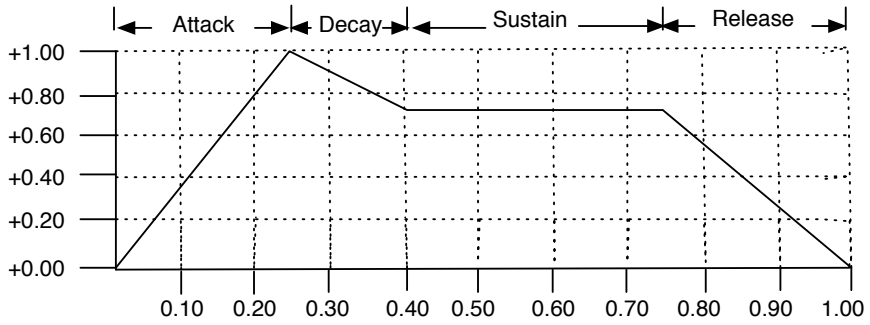


Figure 6.3: A typical ADSR envelope.

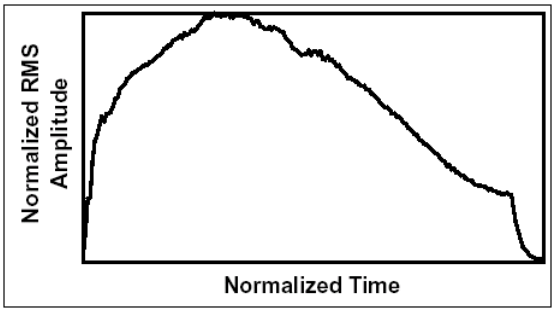


Figure 6.4: A typical trumpet amplitude envelope (a mezzo forte Ab4 from an ascending scale of tongued quarter notes). Figure borrowed from the work of Dannenberg, Pellerin, and Derenyi [11].

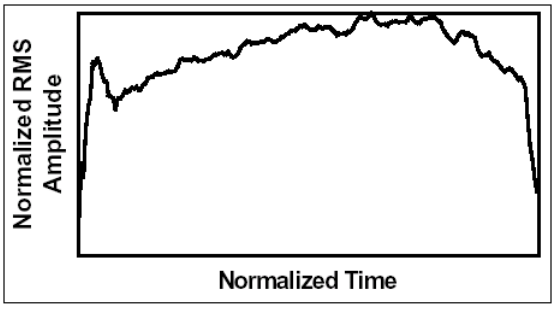


Figure 6.5: A typical trumpet slurred amplitude envelope (a mezzo forte C4 from an ascending scale of slurred quarter notes). Figure borrowed from the work of Dannenberg, Pellerin, and Derenyi [11].

Dannenberg, Pellerin, and Derenyi demonstrated contextual dependencies of measured trumpet envelopes [11]. The comparison between Figure 6.4 and Figure 6.5 show how the amplitude envelope of a note can change drastically under different music contexts.

An optimal shape of synthetic envelope should be as close to the actual acoustic envelope as possible, as illustrated in Figure 6.6. A typical metric for measuring the difference between a synthetic envelope and an actual one is to take the RMS of the value differences between two envelopes at selected time points. Moreover, Horner's work [24] has attempted to characterize how accurate envelopes and spectra should be to be perceptually similar. It is a useful reference of picking the effective error metrics for evaluating synthetic envelopes. As suggested, some good ones are relative-amplitude spectral error, and RMS relative-amplitude spectral error.

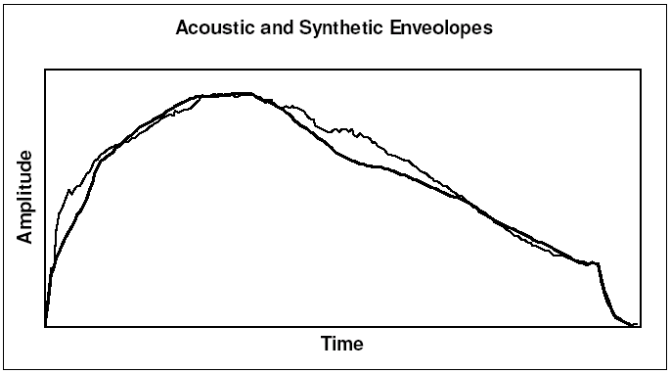


Figure 6.6: An actual amplitude envelope (thin line) and a synthetic envelope (heavy line). Figure borrowed from the work of Dannenberg and Derenyi [11].

To ensure a successful performance model for amplitude envelopes, there are two important and closely related issues to be tackled. One is how to effectively represent envelope shapes, and the other is how to correctly learn the mapping scheme from the information extracted from score (music context) to envelope shapes.

6.2.1 Envelope representation

As seen in Figure 6.4 and Figure 6.5, amplitude envelopes are univariate time series, and there are some specific properties of amplitude envelopes (for a note):

- It is a continuous curve with specific duration.
- The envelope shapes are roughly arched, with two ends lower (not necessary zero) and middle higher in values.
- It is well known that acoustic envelopes can be segmented into several continuous parts similar to the ADSR model, e.g. attack, sustain, decay, release, etc. They are all very important and have distinct characteristics, for example, the duration of attack is always very short (smaller than 30 milliseconds) and has a steep slope change, while the duration of the sustain is mostly smooth and greatly determined by the note duration.
- Listeners are more sensitive to attack and release portions in a note, especially attacks.

According to these amplitude envelope characteristics, we implemented two types of envelope representations. One is to define a 9-point piecewise-linear function to approximate the envelope curve, while the other is classifying shapes into different classes using K-Means. The former can be deemed as an optimization (curve fitting) problem, while the latter is classification. We incorporate domain knowledge in both of them.

6.2.1.1 Curve fitting

Horner and Beauchamp's study on comparing various methods of piecewise-linear approximation of additive synthesis envelopes [23] shows that a Genetic Algorithm (GA) and a sequential enumeration (greedy) method outperform other approaches in finding the appropriate breakpoints in the amplitude envelope of a trumpet tone. However, the curve-fitting problem can be solved exactly in polynomial time, so these heuristics are not necessary [14]. Another interesting conclusion from Horner and Beauchamp's study is that with 12 or more breakpoints in the envelope, and piecewise-linear approximation based on those breakpoints to re-synthesis the envelope, the synthetic tones can be confused with the originals half the time by an average listener.

Their approach of finding breakpoints is simple and general enough - they basically just deem the envelope a curve with arbitrary shape. However, we believe that by incorporating the distinct structural characteristics of amplitude envelopes, a more accurate and elegant representation can be obtained. For example, the attack portion is short and usually has steep slope change while the sustain is long and smooth. Structure is important here because rather than simply storing and recreating envelopes, we want to *generate* envelopes according to rules and parameters. Thus we set out to design a 9-point piecewise linear function $y(x)$ embedding domain knowledge of the envelope. The basic metric is a simple piecewise linear function:

$$y(x) = y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i) \qquad if\ x_i \leq x < x_{i+1} \tag{6.1}$$

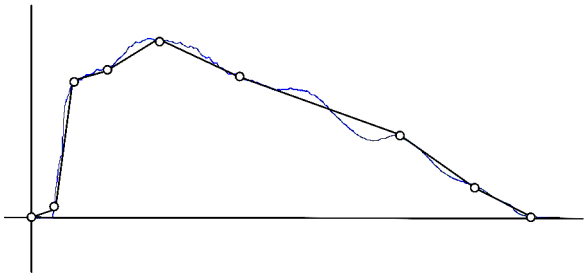


Figure 6.7: The 9-point piecewise linear interpolation function fitted to the amplitude envelope of a note

and the parameters of the functions are a sequence of pairs (x_i, y_i), where $i \in [0, 1, \ldots 8]$. Each pair denotes a "breakpoint" on the fitted piecewise linear envelope, x_i refers to the time, and y_i is the normalized amplitude value. Also there are some constraints imposed on them:

$$x_{i+1} - x_i \geq 10^{-4}, \quad i \in [0, 1, \ldots 7]$$

$$x_2 - x_0 \leq 0.03$$

$$y_2 > y_1$$

$$0.2\,(x_8 - x_0) \leq x_4 \leq 0.8\,(x_8 - x_0)$$

$$y_4 = \max\left(y(x)\right),\ where\ 0.2\,(x_8 - x_0) \leq x \leq 0.8\,(x_8 - x_0)$$

$$\left| \frac{y_{i+1} - y_i}{x_{i+1} - x_i} \Big/ \frac{y_i - y_{i-1}}{x_i - x_{i-1}} - 1 \right| > 0.1,\ i \in [1, 2, \ldots 7]$$

For each note in a piece of music, we extract the amplitude envelope from the segmented audio, set the beginning parameter pairs (x_0, y_0) and the ending parameter pairs (x_8, y_8) to be the beginning and end point of the envelope, and then iteratively fit the function to the envelope. For the sake of simplicity, the envelope starts at time 0, which means $x_0 = 0$. A correctly fitted envelope would look like what is illustrated in Figure 6.7.

The fitting method used is the non-linear least squares regression [60] with the trust-region algorithm [7]. The maximum number of iterations allowed for the fit is set at 400, while the the termination tolerance on the model value is 10^{-6}. Figure 6.8 shows a snippet of the curve fitted result verses the original amplitude envelope. We can see that the curve fitting process

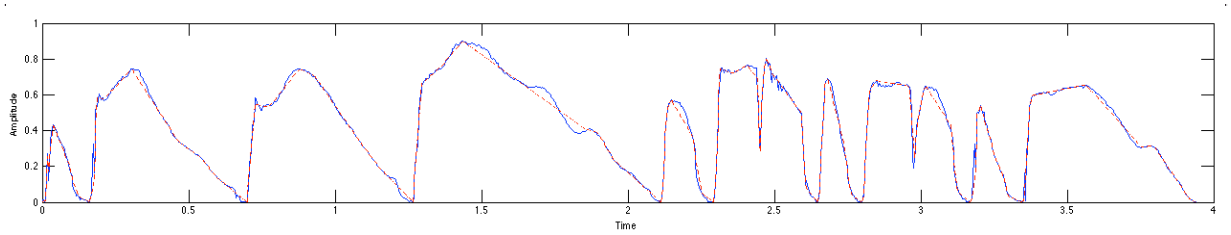


Figure 6.8: A comparison between the original amplitude envelope from the acoustic recording performed by Roger B. Dannenberg, and the fitted curve using the piecewise linear function as in Equation 6.1. The music is the first long phrase in the English Suite Movement V: Finale transcribed by R. Bernard Fitzgerald.

successfully captures most of the dynamics and characteristics of the envelope.

Figure 6.9 shows that even though these two adjacent notes have drastically different arc shapes, the simple piecewise linear function learned from curve fitting represents them without much problem. Of course, some very fine details such as jitter at higher frequencies are lost, but most of them are not audible to listeners anyway.

6.2.1.2 Envelope clustering

Other than fitting a 9-point piecewise linear function to the envelope, we can also deem the digital envelope as a multi-point piecewise linear function with constant interval. Here the interval is set to be 2.9 ms, so that there are 128 samples contained in such an interval if the sample rate of the audio is 44.1KHz.

Contrary to the approach of sampling or concatenative synthesis, we do not intend to store all the amplitude envelopes for synthesis purposes. Instead, we try to find a few representative envelopes that can cover various arc shapes presented in the data. In the envelope construction phase, these are the basic shapes for stretching, compressing and interpolating to generate new envelopes with similar shapes. A natural measure is to cluster the envelopes, and choose or generate one envelope for each cluster to respectively represent all the cluster members. But the original envelopes have various durations. They need to be normalized in duration before they can be properly compared and clustered.

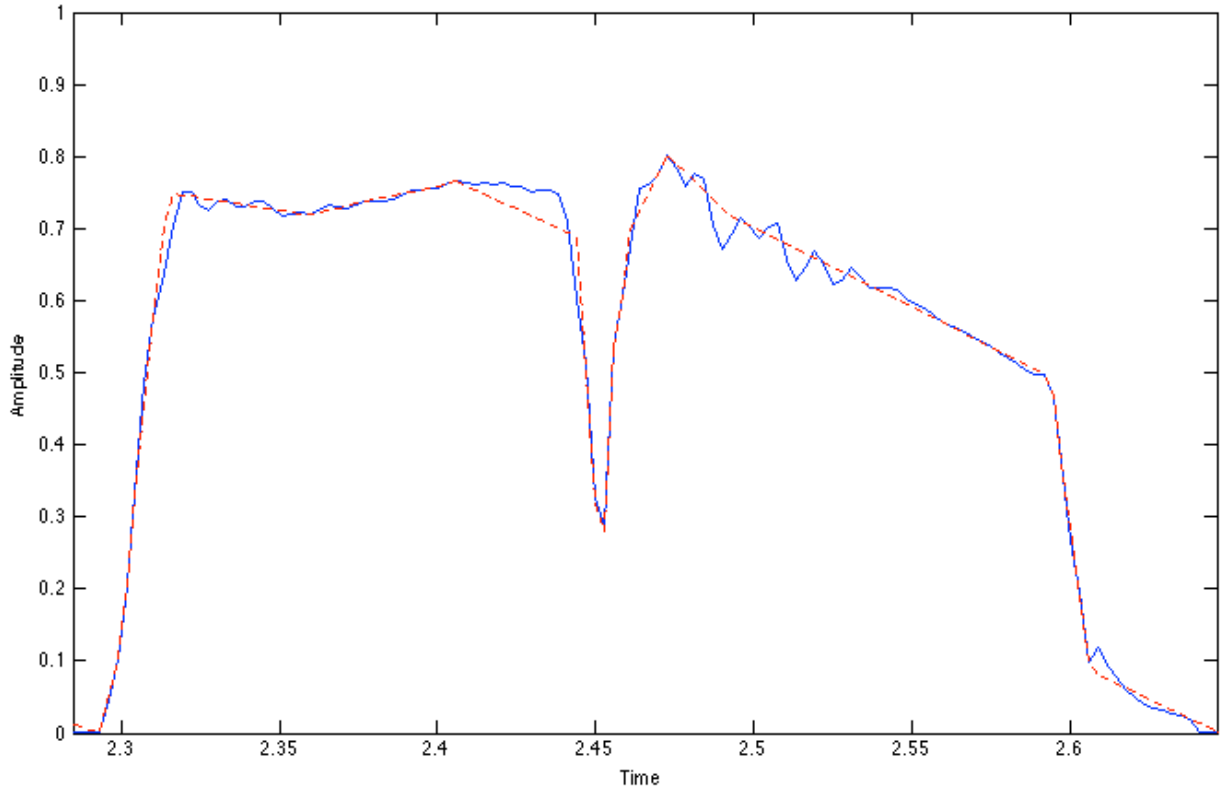


Figure 6.9: A close up look on two adjacent notes (6th and 7th) from Figure 6.8.

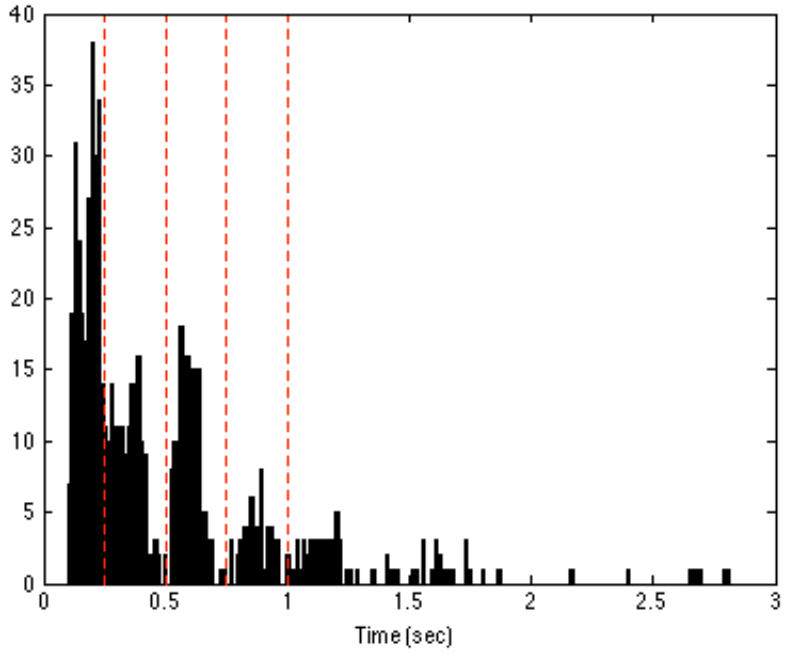


Figure 6.10: Histogram of note duration

As shown in Figure 6.10, duration of notes from the collection ranges from 0.1 to 2.8 seconds. And according to the histogram, they can be be categorized with boundaries set at 0.25, 0.5, 0.75, and 1.0 seconds, as illustrated by the red dashed lines in the figure. So the categories of duration (in seconds) are defined as: (0, 0.25], (0.25, 0.50], (0.5, 0.75], (0.75, 1], and (1, $+\infty$), and the their norm duration are: 0.2, 0.4, 0.6, 0.9, and 1.2 respectively.

Then the amplitude envelope of each note is normalized according to the following rules:

1. Find the duration category a note falls into.
2. $\forall\,(x_i, y_i)$ where $x_i < 0.03$, keep the absolute time value. This keeps the short attack portion unchanged;
3. $\forall\,(x_i, y_i)$ where $x_i > dur_i - 0.03$, keep the absolute time value. This keeps the short release portion unchanged;
4. Linearly scale in time the remaining points to make the note duration the same as the norm duration of the category.

5. Normalize the amplitude of the note to a maximum of 1.
6. When comparing any two normalized envelopes with different norm duration, silence is appended to the shorter one.

The reason for classifying envelopes into different duration categories is simple: the shape of the amplitude envelope is closely related to its duration. Two envelopes with great difference in duration naturally have great discrepancies in perception. It does not make much sense to stretch an amplitude envelop of short duration to simulate a new one with a long duration. Experiments also proved that this normalization step greatly helps achieve good classification results in the later phase of mapping music context to representative envelopes.

K-means, as an important flat clustering algorithm, is chosen for the envelope clustering task. It aims to partition n objects into k mutually exclusive clusters in which each object belongs to the cluster with the nearest mean. The main idea is to define k centroids, one for each cluster. Such centroids can be deemed as the representative envelopes. The algorithm aims to minimize an objective function, in this case an *average squared Euclidean distance* function of objects from their cluster centroids, and each centroid is the mean of the objects in that cluster. The ideal cluster in k-means is a sphere with the centroid as its center of gravity. Also the clusters should not overlap in an ideal situation. It can be deemed as a special case of a Gaussian Mixture Model, with all covariances diagonal, equal, and small.

The algorithm runs according to the following steps:

1. Place k points into the space represented by the object being clustered. These points represent initial cluster centroids.

   Choosing the right initial cluster centroid positions, sometimes known as *seeds*, is essential in avoiding the sometimes poor clusterings found by the standard k-means algorithm. A basic method is to select k observations from the object set at random. In recent years, advanced algorithms were proposed for the purpose, for example, k-means++ [2]. It is an approximation algorithm for the NP-hard k-means problem. With its initialization, the

algorithm is guaranteed to find a solution that is $O(\log k)$ competitive to the optimal k-means solution.

2. Assign each object to the cluster that has the closest centroid.
3. When all data points have been assigned, recalculate the positions of the k centroids. Repeat Steps 2 and 3 until the centroids no longer move. This produces a separation of the objects into clusters from which the objective function can be calculated.

K-means clustering is easy to implement and works with large data sets. It is often used as a pre-processing step in many tasks.

Determining the appropriate number of clusters k can be tricky. The optimal choice of k will strike a balance between maximum compression of the data using a single cluster, and maximum accuracy by assigning each data point to its own cluster. A simple rule of thumb [37] sets the number according to the number of objects n.

$$k \approx \sqrt{\frac{n}{2}} \tag{6.2}$$

Since we classify and normalize the notes according to their duration, it makes sense to run k-means for each category. According to Equation 6.2, the value of k for each category (in the ascending order of category range) should be 11, 9, 8, 5, and 6 respectively. However for proper representation, categories with shorter duration do not necessarily require more clusters than the ones with longer duration, even if they have more training samples. Hence we chose k of all categories to be 6, and there are a total of 30 clusters.

Figure 6.11 shows the envelope shapes at the cluster centroids. As the initial cluster centroids are picked at random, k-means clustering yields a different clustered result every time the algorithm runs. The quality of the clustered result is measured by averaging the distance of all the objects from their cluster centroids. Note that the clustered result chosen is not necessarily the one with lowest average distance (i.e. best quality), but rather the one that yields best classification result of the decision tree model in the phase of mapping music context features to

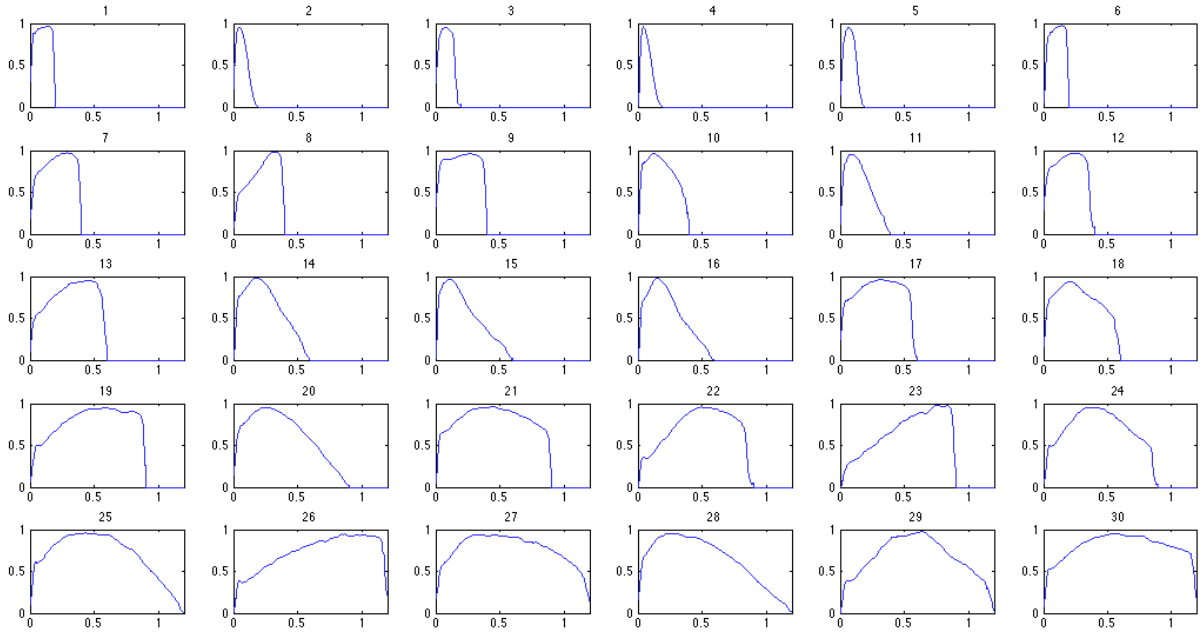


Figure 6.11: Amplitude envelopes at k-means cluster centroids.

representative envelopes.

The representative envelopes are chosen at the cluster centroids, which means they are derived by averaging across all the envelopes in the same cluster. An alternative is to pick the real envelopes that are closest to the cluster centroids. The former approach is preferred, as that smooths out accidental jitters in individual envelopes and only keeps the overall shape.

It should be noted that the curve fitting and the envelope clustering methods are not mutually exclusive. They can be used together to save memory consumption of the model. For example, we can train a greater number of clusters k in k-means, while applying curve fitting on the representative envelope shape at the cluster centroids. This way, the model remains compact as well as easy to manipulate during the envelope construction phase.

6.2.2 Mapping scheme

We now have a collection $O = \{o_1, o_2, \ldots o_n\}$, where n denotes the total number of notes in the collection, and in this trumpet case, 673 note samples in 5 pieces of music. For each note o_i

in the collection O, a feature vector $\vec{c_i}$ and a class label a_i are extracted as previously described. $\vec{c_i}$ represents the music context, while a_i is the envelope cluster that this note belongs to. These are the training and testing data for constructing the model that learns the mapping scheme from feature vectors of music context to envelope cluster labels.

A decision tree model is trained to learn the rules for deciding the envelope cluster label a given music context $\vec{c}$. Decision tree learning is a common machine learning method in data mining [53]. It constructs a predictive model which maps observations about an item to conclusions about the item's target value though explicit tree structures. In these tree structures, leaves represent target values and branches represent conjunctions of features that lead to those target values. The target value could be either nominal (classification tree) or numeric (regression tree), while the branches could be binary splits (splitting two ways) or multiway splits. A binary classification tree is used for the task. We think this is an appropriate model for this case, as acoustic recordings are logic interpretations of music scores, which means the underlying mapping scheme from music context to envelopes can be rule-based.

The learning process of a decision tree is a top-down induction by recursively partitioning a source set into subsets, and repeating on each derived subset, until all the data points in the subset at a node has the same class label or splitting no longer adds value to the predictions. A split criterion needs to be established to judge how well the variable splits the set into homogeneous subsets that have the same value of the target variable. Common split criteria include Gini impurity and information gain. Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it were randomly labeled according to the distribution of labels in the subset. Information gain is based on the concept of entropy used in information theory [41], and turns out to be a good split criterion for the task.

For a random variable X with n outcomes $x_1, \ldots, x_n$, information entropy is a measure of uncertainty defined as:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_b p(x_i)$$

Here $p(x_i)$ is the probability mass function of outcome x_i. b is the base of the logarithm used. Common values of b are 2, Euler's number e, and 10. We use the binary entropy function in this case, i.e., $b = 2$. The binary entropy can be understood as the expected number of bits needed to encode a randomly drawn value of X.

In general terms, the expected information gain for an attribute A on a set of training examples S is the change in information entropy from a prior state to a state due to sorting on A:

$$IG(S, A) \equiv H(S) - \sum_{v \in values(A)} \frac{|S_v|}{|S|} \cdot H(S_v)$$

where $v \in values(A)$ is a value of A.

In addition, a cost function $Cost(i, j)$ needs to be defined. It denotes the cost of classifying a point into class j if its true class is i. A common cost function for a classification tree is:

$$Cost(i, j) = \begin{cases} 1, & if\ i \neq j \\ 0, & if\ i = j \end{cases}$$

Instead we define the cost function as follows in order to take into account the actual distances among cluster centroids:

$$Cost(i, j) = \begin{cases} d(C_i, C_j)^2, & if\ i \neq j \\ 0, & if\ i = j \end{cases}$$

Here $d(C_i, C_j)^2$ is the squared Euclidean distance between two cluster centroids C_i and C_j.

There are several distinct advantages of using a decision tree model:

- It is simple to understand and interpret.
- It requires relatively little effort for data preparation. Other methods such as regression

models require specific generalization and normalization on the data, while decision trees implicitly perform feature selection, are not sensitive to outliers, and can work with data with either missing values or nonlinear relationships among variables.

- It is a white-box model. Logical expressions or rules can be easily induced from a learned decision tree.

However, in practice a fully grown decision tree often needs to be pruned, otherwise it can overfit the training data and perform badly on the actual data when the distribution of the training data is different from the actual data distribution. Pruning optimizes tree depth by merging leaves on the same tree branch. It aims to reduce the size of a learning tree without reducing predictive accuracy as measured by a test set or using cross-validation. An optimal pruning level is found by minimizing test set loss through scanning the fully grown tree from the bottom up. Another pruning factor is the minimum number of observations per tree leaf $MinLeaf$. It is determined by measuring classification errors on the training and test data.

When we synthesize one out of five pieces of music recorded for comparison, the data extracted from the one being synthesized is the test data, while the rest are training data. This could be deemed as an approximation of 5-fold validation on the decision tree learning. Figure 6.12 shows a decision tree learned based on the k-means clusters as illustrated in Figure 6.11 and with <English Suite Movement V: Finale> as the test data. Variables for split conditions in the tree are: *note_dur* (note duration), *sil_pre* (duration of silence before the note), *sil_post* (duration of silence after the note), *slur* (value 0/1/2/3 for not in/beginning/middle/end of a slur), *dur/ioi_pre* (ratio of note duration and previous note's IOI), *accent* (whether the note has an accent mark), *vel* (note velocity), *vel/vel_post* (velocity ratio between the note and the previous one), *vel/vel_post* (velocity ratio between the note and the previous one), *note_ioi* (note IOI), and *pitch_diff_pre* (pitch difference between the note and the previous one). It has $MinLeaf$ set at 7 and is pruned at level 15 (out of a total of 35 levels when fully grown). The classification errors of the training and test set for this particular tree are 1.486036 and 0.931697 respectively.

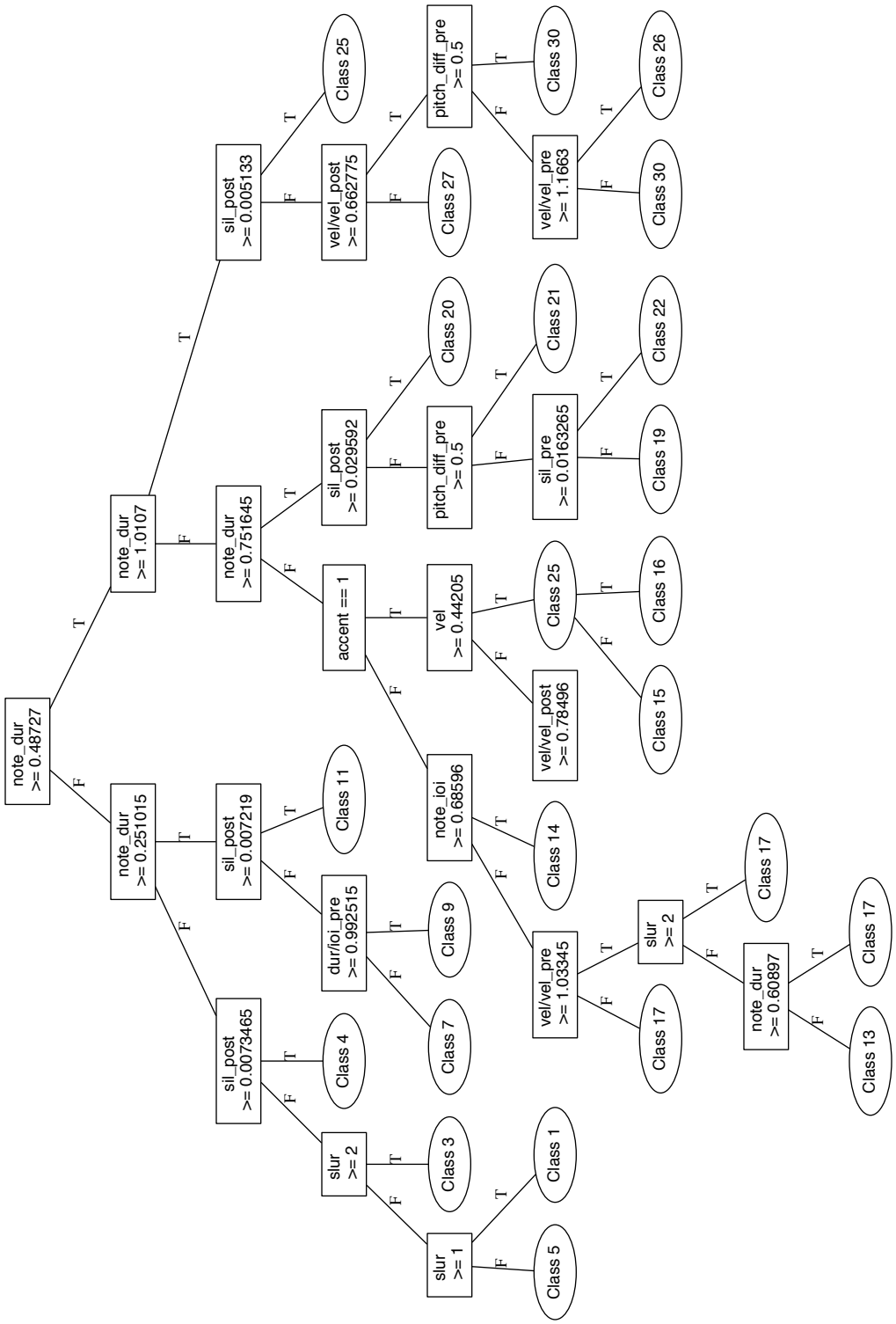


Figure 6.12: A learned decision tree structure graph.

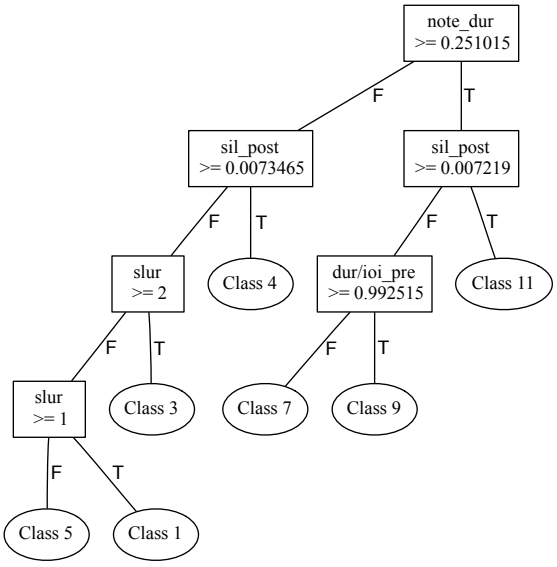


Figure 6.13: A close-up look on a portion of the decision tree in Figure 6.12.

If we observe the structure of the learned decision tree as in Figure 6.12 from the top down, we can see that the first two levels are mainly splitting on *note_dur* (note duration), which are well in line with the category boundaries in envelope normalization. When further examining the subtree with the root splitting at 0.25 second of note duration, as shown in Figure 6.13, we can see that *sil_post* (duration of silence after the note) is used for splitting, probably to distinguish whether the amplitude value should reduce to 0 towards the end of the note. Then branches are splitting along different values of *slur* (value 0/1/2/3 for not in/beginning/middle/end of a slur), and *dur/ioi_pre* (ratio of note duration and previous note's IOI) before reaching the leaf nodes. Overall many parts of the tree structure can be logically explained, though the tree itself is purely learned from the data without human intervention.

The learned decision tree is the core of the performance model for generating amplitude envelopes. Given the music context of a note extracted from the score represented in MIDI and MusicXML, the decision tree is able to output the representative envelope used for interpolation to synthesize a new envelope.

6.2.3 Envelope interpolation

The learned decision tree classifies a note with a set of contextual variables extracted from the musical score to an envelope class. The curve-fitted parameters corresponding to the representative envelope of the class is used as the basis shape for generating the new envelope. The interpolation process is quite straightforward and similar to the envelope normalization step in the envelope clustering module:

- For the attack portion, i.e. for all the points (x_i, y_i) where $x_i < 0.03$, keep their relative time invariant;
- For the release portion, i.e. for all the points (x_i, y_i) where $x_i > dur - 0.03$, keep their relative time invariant;
- Scale the rest of the envelope along the time axis to make the envelope duration equal to the duration of the note;
- Set y_0 of the first breakpoint to be the last amplitude value from the previous note if it does not end with silence;
- Others are scaled by the relative velocity of the note;
- Use piecewise linear function to generate samples in between breakpoints.

If the envelope interpolation is based on the 9-point piecewise linear representation from curve fitting, and a smoother curve is desired, a piecewise cubic Hermite interpolation [19] can be used. Figure 6.14 shows the comparison between the synthesized envelope using piecewise linear interpolation, and the one using piecewise cubic Hermite interpolation. The latter has noticeably smoother curve than the former. However, often times the differences are not audible to average listeners.

In summary, the performance model for amplitude envelopes consists of a feature extraction module to acquire contextual parameters from MIDI, a learned decision tree for mapping contextual parameters to representative envelope class, and an interpolation process that transforms

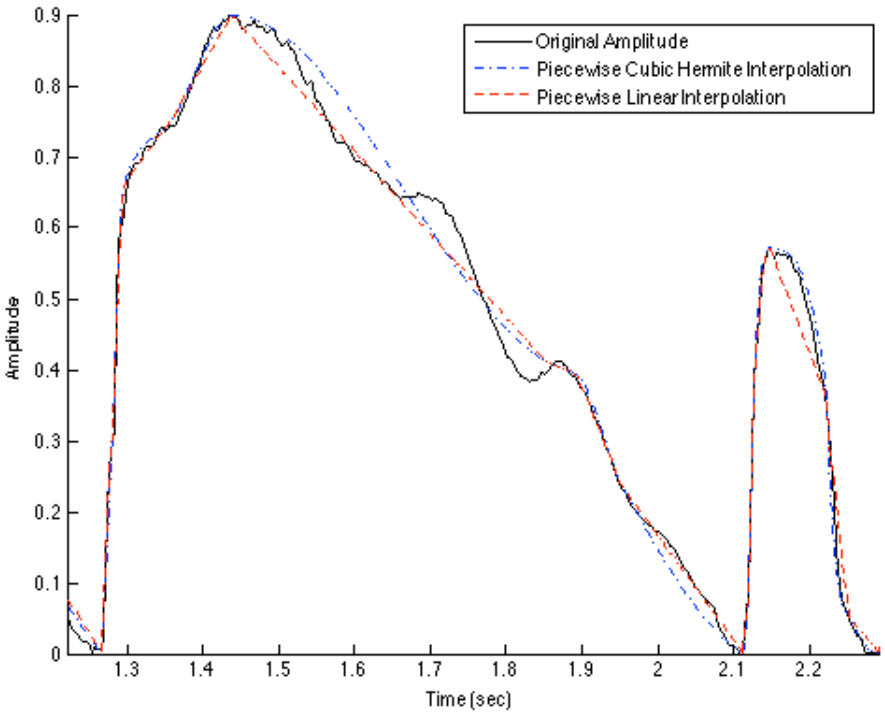


Figure 6.14: Comparison between different interpolation methods.

a representative envelope to a desired new one.

6.3 Frequency Envelope

Frequency envelopes refer to the continuous F0 frequency contour for notes. They are also very important in synthesizing realistic instrument sounds. A synthetic performance with steady, unwavering frequencies sounds artificial when compared to a real recording. Frequency envelopes possess different characteristics from amplitude envelopes. The main differences lie in the envelope shapes. Amplitude envelopes are mostly arch shaped and can be approximated with a piecewise linear function interpolating among breakpoints, but frequency envelopes contain transitions from the previous note and to the next note, plus local fluctuations for vibrato and instability. All those details are essential for synthesizing a realistic frequency envelope.

In speech-to-singing synthesis work, Saitou, et al. [63] proposed a set of hand-tuned func-

tions for modeling F0 frequency contour. This is based on the identification of four types of F0 fluctuations: overshoot, vibrato, preparation, and fine fluctuation. They are defined as follows:

1. Overshoot: a deflection exceeding the target note after a note change.
2. Vibrato: a quasi-periodic frequency modulation (4-7 Hz).
3. Preparation: a deflection in the direction opposite to a note change observed just before the note change.
4. Fine fluctuation: an irregular frequency fluctuation higher than 10 Hz.

Overshoot and preparation are global F0 changes that correspond to the music context, while local F0 changes include vibrato and fine fluctuation. The parameter values in the function were determined using the nonlinear least-squared-error method to minimize errors between the generated F0 contours and actual ones. The authors reported that "the proposed system can convert speaking voices into singing voices whose naturalness is almost the same as actual singing voices."

We also use hand-tuned function to model overshoot, preparation and vibrato, and introduce white noise to simulate fine fluctuation. The listening test shows that such a simple approach can generate realistic instrument sounds.

6.3.1 Synthesizing overshoot, preparation and vibrato

We define a set of breakpoints and use piecewise cubic Hermit interpolation to generate overshoot and preparation curves. Vibrato is generated using an oscillation model with increasing amplitude. Compared with the second-order non-linear functions that Saitou, et al. defined in their paper[63], our method is much easier to tweak, and still synthesizes similar curves.

The frequency contour of each note may consist of three consecutive portions, first overshoot, then vibrato, and the last preparation. The duration of overshoot and preparation are set as:

$$dur_{overshoot} = \min\left(35ms, \frac{dur_{note}}{2}\right)$$
$$dur_{preparation} = \min\left(40ms, \frac{dur_{note}}{2}\right)$$

And the rest is vibrato. Not all the notes have these three portions though. Overshoot and preparation should exist only for those notes transitioning from one to the other. That is to say, overshoot will appear only when a note is at the middle or end of a slur, and there is pitch difference between the previous note and itself; while preparation will appear only when a note is at the beginning or middle or a slur, and there is pitch difference between the next note and itself.

Four breakpoints (x_i, y_i) are to be defined for an overshoot curve. They are:

1. $x_1 = 0$, $y_1 = \frac{F0_{n-1} - F0_n}{2}$. $F0_{n-1}$ and $F0_n$ denote the fundamental frequency (or pitch in Hertz) of the previous and current note;
2. $x_2 = \frac{1}{2} Dur_{overshoot}$, $y_2 = 0$;
3. $x_3 = \frac{3}{4} Dur_{overshoot}$, $y_3 = -sign\left(F0_{n-1} - F0_n\right) \cdot \min\left(w_o \left|F0_{n-1} - F0_n\right|, F_{cap}\right)$, where w and F_{cap} are predefined constants controlling the deflection shape, here $w_o = 0.1$ and $F_{cap} = 7$.
4. $x_4 = Dur_{overshoot}$, $y_4 = 0$.

The definition of breakpoints for a preparation curve is very similar, just with a reverse shape, as shown in Figure 6.15.

Vibrato for longer notes is usually more perceptible, thus we design the synthesizing function for vibrato to be an oscillator model with increasing volume as below:

$$h\left(t\right) = F_{mv} \cdot \max\left(w_i t, 1\right) \cdot \sin\left(w_v t\right)$$

Here F_{mv} denotes the maximum frequency variation by vibrato, w_i controls how fast the increasing volume is, and w_v determines the vibrato frequency. In the case of trumpet $F_{mv} = 5$, $w_i = 1$,

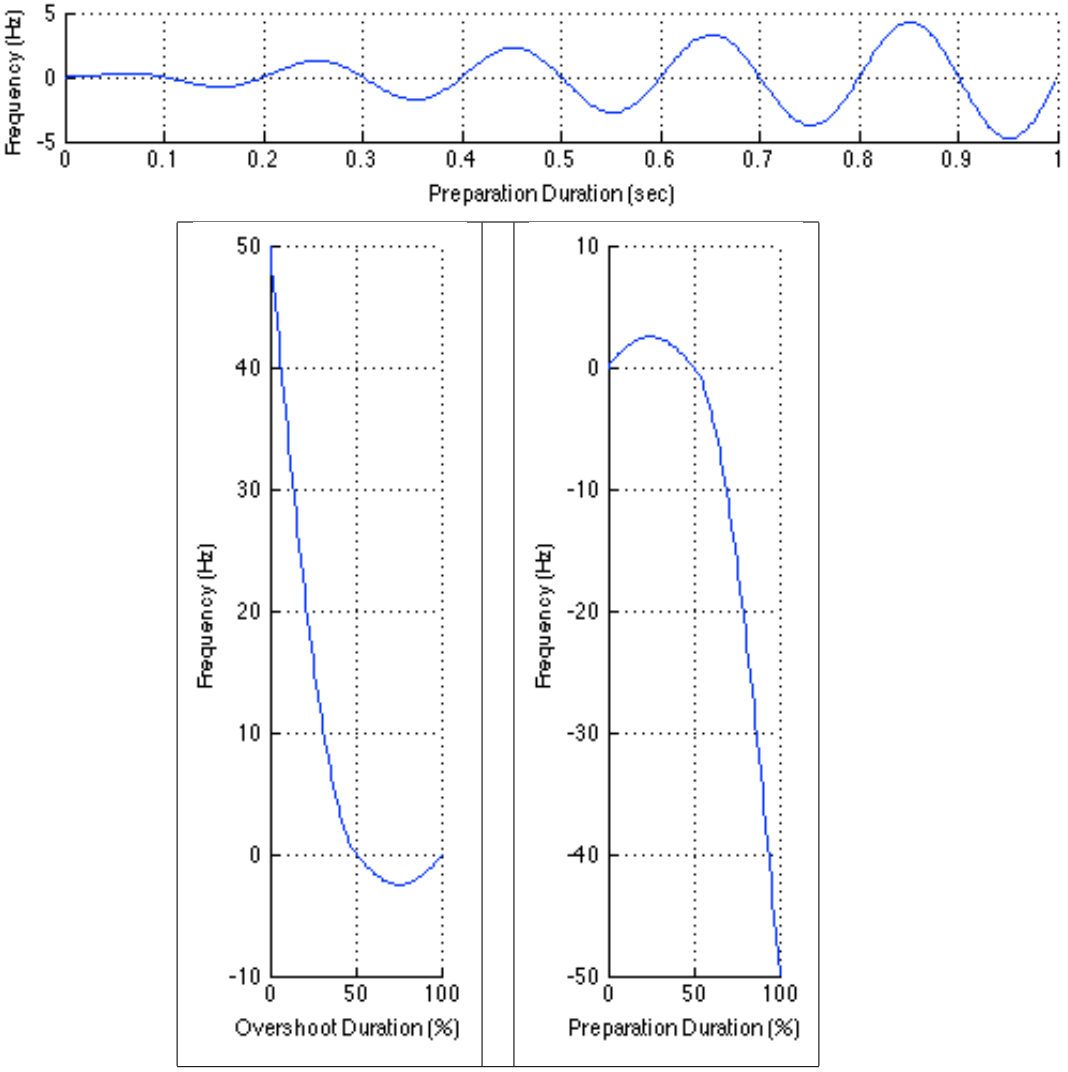


Figure 6.15: Synthesized overshoot, vibrato and preparation curves.

and $w_v = 10\pi$, which means the vibrato frequency is set at 5Hz. A synthesized vibrato example is shown in Figure 6.15.

6.3.2 Adding fine fluctuation

Fine fluctuation is introduced by adding white noise to the frequency contour.

A white Gaussian noise is added to the frequency envelope as the fine fluctuation. For easy computation, the sample interval of the frequency envelope is 2.9 ms, the same as that of the amplitude envelope. The noise in Hertz consists of a sequence of pseudorandom values drawn from the standard normal distribution. Its effect is linearly scaled by R_f. For trumpet R_f is set at 0.2. Note that probably it would have been better to make the noise proportional to frequency, so that the frequency deviation is smaller when the pitch is lower and with smaller semitone difference in frequency. On the other hand, trumpet frequency in the lower octave could be less stable than upper octaves, so a model with a fixed noise amplitude at any frequency might be a good approximation. Since the sound synthesized by the current model is deemed realistic enough, we leave further investigation on this matter for future research.

Figure 6.16 shows a comparison between the actual frequency envelope and a synthesized one for a snippet of the <English Suite Movement V: Finale>. They look similar in terms of fine characteristics, though the actual frequency envelope has more noise especially at the beginning and end of notes due to pitch detection error. Listening tests also show that the synthesized frequency envelope, combined with the synthesized amplitude envelope, is able to generate realistic instrument sounds.

6.4 Previous Attempts

As the goal of this thesis is to apply appropriate machine learning techniques to replace hand-tuned functions in constructing a high-quality musical instrument synthesizer, finding the right

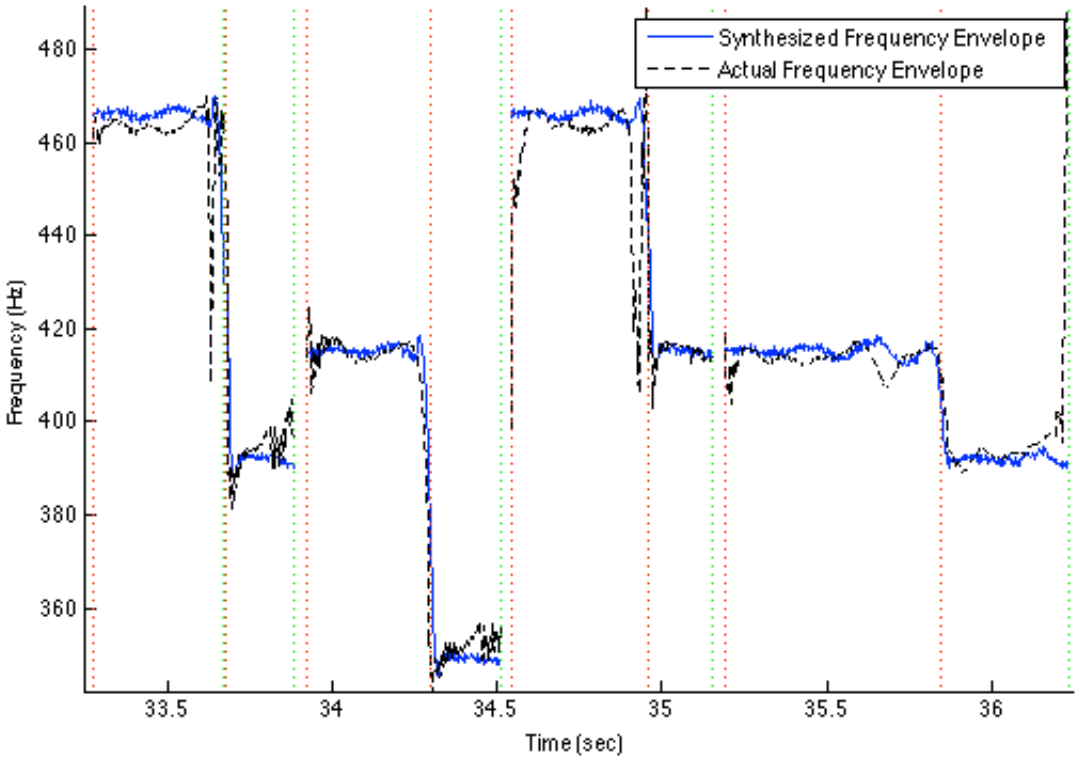


Figure 6.16: Comparison between the actual frequency envelope and a synthesized one.

machine learning model that can properly learn the mapping scheme between music context and sound envelopes is the essential problem. Since our research effort on the subject has been on and off for a long time, we have tried quite a few approaches with different models over the past several years. They include some machine learning models for synthesizing amplitude envelopes, namely the neural network, the nearest neighbor, and the Hidden Markov Model. We also tried to build a learning model for frequency envelopes.

6.4.1 The neural network approach

Due to the success of using a neural network model for the task of accurately detecting note onset and segmenting audio, which is also a significant contribution of this thesis, we started out trying to construct the performance model with a neural network. We were hoping that the neural network could directly map the music context signals to the a set of parameters representing the envelope curve. Thus the designed neural network is a multi-layer feed-forward network with multiple outputs. However, we quickly realized that this approach did not work, as a neural network with simple inner structure (1~2 layers) does not suffice, while a more complex model with 3 or more layers just will not converge during training.

Below are some possible reasons to explain why the neural network approach did not work based on our analysis:

1. Problem complexity: our previous successful experience of using neural networks was to produce a binary response based on instantaneous and rich audio features, and it was bootstrapped with hierarchical information from audio-to-score alignment in probability form. In contrast, the performance model needs to synthesize a complex time-series curve from a set of abstract score context parameters, which may or may not be accurately and consistently expressed in the training samples. Basically the model needs to learn the context of music, i.e., high-level concepts, which is a far more complex problem than the instantaneous note onset detection.

2. Limited quantity of training samples: lacking a big pool of training data is the constraint of this thesis research ever since its initialization. For a complex learning problem like this, a more sophisticated model may still yield good result with abundant training data. Recent progress on deep learning [4] also shows that it is now possible to train a deep neural network to learn complex feature representations or high-level concepts. But with merely 5 pieces of music and a total of 673 notes to start with, while at least 1/5 of the data needs to be used for testing and evaluation, and only the remaining 4/5 can really be used for training, we just do not have enough data to successfully train a complex model. One of our goals is to acquire synthesis and performance models on the basis of a short recording session with an expert musician, so using a much larger data set is not an option.

6.4.2 The nearest neighbor approach

The nearest neighbor approach was inspired by the concatenative synthesis and the software Synful as described in earlier sections 2.1.1.4 and 2.1.2.1. Basically the system stores all the sound samples in a database, finds a note sample with similar context information to the one to be synthesized, and uses its corresponding envelope plus some simple stretching and tweaking, as the output of the performance model. This approach worked generally well, and was able to synthesize sounds that are quite realistic. However, it was eventually abandoned for the following reasons:

1. Its underlying principle is not that much different from the concatenative synthesis or even sampling synthesis with multi-sampling. It certainly has a negative impact on the significance or novelty of this research.
2. All the sound envelopes have to be stored in memory. To save memory consumption, we started using curve fitting to greatly reduce the number of parameters needed to represent an envelope.
3. The distance function of the nearest neighbor model is designed by hand. The advantage

of such a model is that it does not require training, but of course the consequence is it is rather arbitrary to determine whether two notes are similar or not.

6.4.3 The hidden Markov model approach

A significant recent development in the research field of speech and singing voice synthesis is the Hidden Markov Model approach, as described in related work sections 2.2.3 and 2.2.4.1. It proved to be very successful and is able to produce natural sounding synthesized speech samples. However, our effort in constructing such a model for this particular application has not been very successful. Maybe part of the reason is (again) due to the very limited number of labeled samples to work with. Another possible concern is that music is very unforgiving of artifacts. In speech, one can often get away with high intelligibility, but in music it is all about sound rather than meaning, so the synthesized sound has to be extremely good. This is why in the music synthesis field, sampling has dominated more abstract synthesis methods.

Nevertheless, hopefully in the near future we will be able to see HMM being applied to construct musical instrument synthesis, and compared with the approach described in this thesis.

6.4.4 Learning frequency envelope

It has been proven that hand-tuned functions work very well for synthesizing frequency envelopes. That is probably because the characteristics of frequency envelopes are rather simple. Nevertheless, we were still planning to learn the frequency envelope once we got satisfactory results from learning amplitude envelopes.

We tried constructing the curve descriptive function to be either second-order functions similar to what Saitou et al. designed [63], or a set of piecewise cubic Hermit spline functions like the hand-tuned functions that are currently used in the system 6.3.1. Then we tried learning the parameters of the curve descriptive function using the machine learning techniques similar to that of amplitude envelope learning. Neither was very successful. One main reason is too much

accidental error exists in the detected pitch contour, especially at the beginning and end of a note. That is where the two important components, overshoot and preparation reside. All those pitch detection errors are enough to prevent an overshoot or preparation model from being successfully learned. Eventually we gave up on the machine learning approach for frequency envelope and switched back to the hand-tuned function.

Of all the models that we built and tried for the performance model, the current k-means plus decision tree approach stands out to to be most systematic and elegant. It is general enough that other instrument synthesis can be easily modeled with this approach. It also produces the most realistic synthesized sound.

6.5 Summary

In this chapter, we demonstrated that by utilizing optimization and machine learning techniques, a performance model that generates amplitude and frequency envelopes given a music score can be automatically constructed. As the process generalizes quite well, we can easily adopt the model for synthesizing musical instruments other than the trumpet.

Chapter 7

Modeling Other Instruments

After the trumpet synthesis is automatically constructed with a set of acoustic recordings and corresponding scores, naturally the next step is to model other instruments. This is an important verification step to see whether the built infrastructure can generalize well.

To put the built system to the test, we chose a set of acoustic recordings for bassoon performed by Mark Dalrymple. It consists of 6 pieces of music; all are famous bassoon tunes. They include:

- P. Dukas - The Sorcerer's Apprentice, 1st Part
- Tchaikovsky - Symphony No. 4 in F Minor, Op. 36, Fagott 1, Main Theme
- I. Strawinsky - The Rite of Spring, 1st Part
- S. Prokofiev - Peter and the Wolf, Grandpa Theme
- I. Strawinsky - The Firebird, Berceuse
- Emerald

By running through the system to build bassoon synthesis and analyzing the results, we have the following observations about the distinct characteristics of the instrument as well as the acoustic recordings, and the consequential effect on the work required to process such data.

1. Bassoon has a wide pitch range, extending more than 3 octaves. The instrument model is built to synthesize tones with pitch from $B\flat 1$ (MIDI pitch value 34) to $D5$ (MIDI pitch

value 74). Even in a single piece of music, for example, the short first part of the Sorcerer's Apprentice, pitches can range from $C2$ to $D\flat4$, which is more than two octaves.

2. The performances contain many deviations from "ideal" pure tones with rapid onsets. There is some asynchrony among pads opening and closing tone holes that is evident in the sound at note transitions, pervasive breath and mechanical noise, and other inconsistencies in the sounds. Even the single note recordings for constructing the instrument model have some issues with unsteady amplitude or pitch. This along with the broad pitch range of bassoon caused significant overhead in data processing. We implemented some algorithms to correct most of the errors introduced by the audio analysis as described later in this chapter. Still, some parts had to be manually cleaned up where even the analysis error correction module cannot handle the data well.

3. Vibrato in bassoon tones is more noticeable, at least in the recordings being analyzed. Figure 7.1 illustrates the obvious vibrato in a note. Vibrato affects not only frequency contour, but also amplitude envelope. This caused some concerns as the curve fitting function for the amplitude envelope is a piecewise linear metric. However, listening tests show that the synthesized tone sounds just fine, as long as the vibrato is accurately present in the frequency envelope.

4. The fine fluctuations in frequency envelopes are less than that in trumpet envelopes, which means the frequency contour of a bassoon sound is smoother. Thus we turned down the scalar R_f when introducing white noise to frequency envelopes.

Listening tests proved that the synthesized tone for the bassoon sounds realistic. However, the significant overhead for data clean up clearly shows the importance of having very clean performances to begin with, or having a robust procedure to handle noisy data. The extent to which performances should be "clean" and free of noises and inconsistent amplitude or frequency fluctuations is largely a question of taste. Certainly, in this era of highly produced recordings, we are accustomed to having ultra-clean performances, and most players strive for this kind

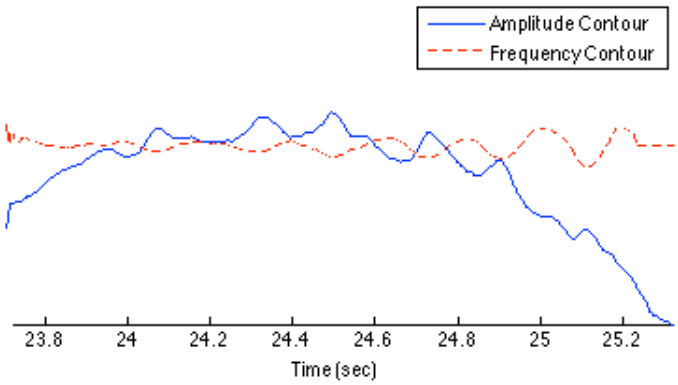


Figure 7.1: Amplitude and frequency envelope of a vibrato bassoon note

of sound. Highly consistent sounds are much easier to synthesize, so the clean output of our synthesizer can be considered either a feature or a limitation, depending upon one's objectives.

7.1 Dealing with Audio Analysis Error

As mentioned in chapter 5, the algorithm used for extracting F0 frequency and amplitude contours from a piece of music is the McAulay-Quatieri partial-tracking method [39] provided by the software SNDAN. It is very successful at tracking wide changes in pitch, whether of the fundamental, or of partial tones. However, due to the limitation of the algorithm itself, the tracking process will stumble where partials peaks are ambiguous, and the program requires that partial tracks to not overlap. So this analysis model does not work well with those sounds with significant amounts of noise. This issue became prominent when we were processing the bassoon samples, and both amplitude and frequency contours were affected. It threatened the quality of the performance model, since if the training data is noisy, no good result will come out of the learning process.

We investigated other types of pitch tracking algorithms and programs. Tolonen et al. proposed a multi-pitch analysis model [64], which does indeed deliver reliable pitch contour. However, the algorithm is using a frame length of 46.4 ms and a hop factor of 10 ms, which is 16 times

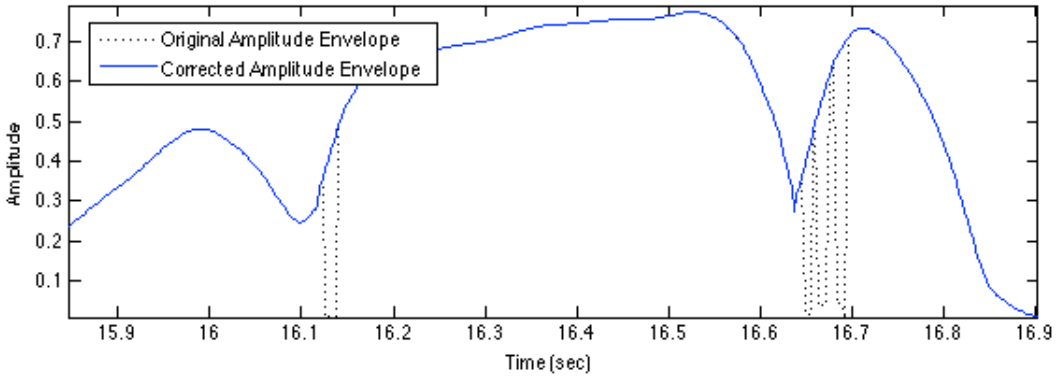


Figure 7.2: Effect of amplitude detection error correction

the analysis frame size of SNDAN, and even larger than the duration of an attack phase (30ms). The algorithm performs significantly worse when the frame size is smaller. Hence our conclusion is that Tolonen's model may work well for high level analysis such as music information retrieval, but it is too coarse for our synthesis purpose.

As we decided to stick with the software SNDAN for audio analysis, we implemented simple algorithms to correct the tracking errors with the help of local context information extracted from the analyzed audio.

7.1.1 Amplitude detection error correction

When an audio frame contains more than one partial track, the McAulay-Quatieri partial-tracking algorithm gets confused, and may think of it as an "empty frame". However, the frames close by probably contain the correct amplitude information. Thus we designed a hill climbing method to detect abnormalities in the amplitude contour, find the two points that are close by and "normal," and interpolate between these two points to fix the abnormal sections. Figure 7.2 shows the correction result on a snippet of <P. Dukas - The Sorcerer's Apprentice, 1st Part>. It removes the undesired artifacts introduced by the audio analysis program, and prepares the data for model learning.

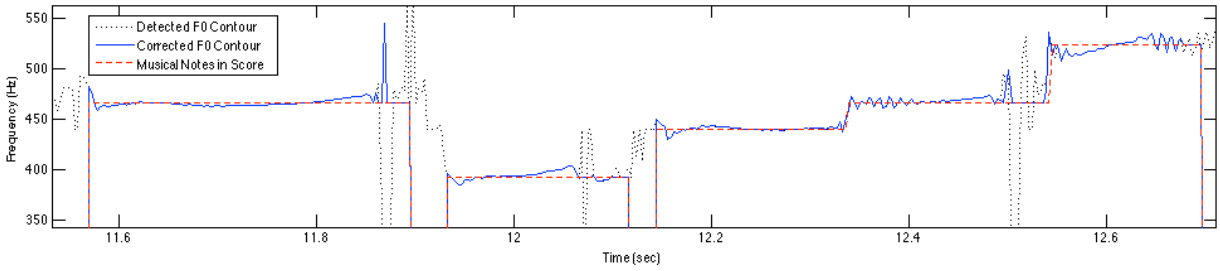


Figure 7.3: Effect of pitch detection error correction

7.1.2 Pitch detection error correction

Like most pitch detection algorithms, the McAulay-Quatieri partial-tracking method tends to make pitch detection errors when the amplitude level is low. A common error encountered is the octave error, which means the pitch is detected one octave too high or too low. Also the analysis frequency (i.e. analysis fundamental frequency) of the program affects the outcome as well. In places where frequency is either not detected or out of bounds, the program outputs the analysis frequency instead of 0.

Fortunately, besides the acoustic recording, we also have the corresponding music score to work with. After accurate audio-to-score alignment and segmentation, we design a simple yet effective rule-based method to correct pitch detection errors. Figure 7.3 shows the comparison among an actual F0 frequency contour, a corrected one, and the corresponding musical notes in the score. Most arbitrary and undesired jitter is removed from the contour after the correction process. It can be further combined with the measure that corrects amplitude detection errors as described earlier to produce smoother pitch and amplitude contours for later processing.

7.2 Summary

The success of constructing synthesis models for both trumpet and bassoon demonstrates that the described scheme generalizes well, and we should be able to model more wind instruments with similar measures. Other types of musical instruments may require significant tweaking or

new developments. For example, the bow transitions of string instruments, like the violin, can be tricky to model. Furthermore, the frequency jitter in violin tones is significant, rapid, and critical for a natural sound. However, we believe that the overall infrastructure can accommodate a broad range of situations.

Chapter 8

Conclusion

8.1 Summary

This thesis research proposes a systematic scheme for automatically constructing high-quality musical instrument synthesis based on a set of performance examples and corresponding musical scores. Besides the traditional synthesis techniques, machine learning and optimization methods play a crucial role. They are used pervasively throughout the whole system in order to automate the data processing and modeling procedures, and intelligently control the synthesis.

The construction process starts with a set of performance examples and their corresponding scores. To generate labeled data for training the performance model later on, we use audio-to-score alignment and accurate note segmentation to acquire the exact correspondence in time between the audio and the score. The alignment is done by finding the optimal path in the similarity matrix of two sequences of chroma vectors, one from the audio, and the other from the score. The result tells us the approximate note boundaries, and is used to further improve the note segmentation. A note segmenter is then trained by running an iterative bootstrapping process until it converges. This way, the resulting precision of the note onset detection is greatly improved to be in the order of milliseconds. Ultimately the alignment and segmentation process detects accurate note boundaries in the audio and their exact correspondence to specific notes in

the score. They are used as the training data for constructing the performance model.

For training an amplitude performance model, we first extract amplitude contours from the analysis of performance examples. They are segmented according to the note boundaries found the audio-to-score alignment and note segmentation process, and then normalized to amplitude envelopes according to a set of duration categories. K-means clustering is run on each category to find the envelope clusters and representative envelopes at the cluster centroids. After that, a decision tree is learned from the the training data, which are pairs of note context feature vector and the envelope cluster its corresponding normalized envelope falls into. At the synthesis process, the decision tree is able to pick a representative envelope shape given context of a note. A new amplitude envelope is interpolated from the representative envelope and used by the instrument model. Meanwhile, frequency envelopes are generated by a set of hand-tuned functions that simulate different types of frequency fluctuation in a note: overshoot, vibrato, preparation and fine fluctuations.

The instrument model is essentially a Spectral Interpolation Synthesis. It consists of two parts, the harmonic model and the attack model. The harmonic model is based on a database of spectra at specific amplitude and frequency thresholds. They are extracted from a set of acoustic recordings of single notes with crescendo. Given the control signals of amplitude and frequency, the corresponding instantaneous spectrum is generated from looking up and interpolating among four nearest spectrum samples in the database. A wavetable is computed from the generated spectrum by a simple harmonic additive synthesis. To increase computational efficiency, we only compute 20 wavetables per second, and interpolate between two tables to generate every sample in between. Pre-recorded attacks are spliced by the attack model at the tongued note onsets. To avoid phase canceling, the phase information at the end of the spliced attack determines all she subsequent harmonic phases within the phrase.

As a showcase of the framework, a high-quality trumpet synthesizer is constructed by learning from a limited number of performance examples and corresponding scores. The framework is then extended to model a bassoon synthesizer.

8.2 Contributions

This thesis makes significant contributions in the field of computer music synthesis. The impact can be considered from three major aspects.

1. The thesis proposes a novel approach of bootstrapping a note segmenter, and improves the precision by an order of magnitude.

   There are many ways of training an accurate note segmenter, yet almost all of them require abundant training data that are manually labeled. We demonstrate that it is entirely feasible to train a note segmenter when there is essentially no labeled data to begin with. By deploying an intelligent bootstrapping method, the segmenter learned over iterations can accurately detect note onsets with a precision on the order of milliseconds.

   Not having enough labeled data for training purposes was one the constraints the research had to face ever since its initialization. However, this constraint essentially led to the effort of building a model for accurate audio-to-score alignment and segmentation, and a side product of bootstrapping a note segmenter. The approach not only generates segmented audio and score pairs as labeled data for this research, but also benefits related research fields such as music information retrieval and score following.

2. The thesis demonstrates that a performance model can be intelligently learned from performance examples and corresponding scores.

   Most music synthesizers refer to merely the instrument model, which generates synthetic audio only if appropriate control signals are given. Thus for the acoustic instruments that are driven by continuous and complex control signals, traditional synthesis approaches can hardly achieve very realistic simulation results. In our belief, the secret of synthesizing convincing sounds of those acoustic instruments, such as the trumpet, lies in the performance model.

   The performance model accepts a musical score as the input, and emits proper control sig-

nals of amplitude and frequency to drive the instrument model, in other words, a typical music synthesis. Previously, the performance model had to be constructed manually and hand-tuned. Now we propose an approach that can automate the process without much manual intervention. What is more, the representative envelopes found by k-means clustering, and the rules learned by decision tree for mapping the musical context to envelope classes are all easy to understand and interpret logically. The solution is not only simple and elegant, but also working well in practice.

3. This thesis describes a complete system framework of a synthesizer synthesizer. It includes not only a demonstration of how to build a good musical instrument synthesizer, but also also an intelligent approach of how to learn to control the synthesizer.

   The framework is a good example of machine learning applications, as various machine learning methods are used throughout the system. It demonstrates that appropriate machine learning techniques can effectively replace the tedious work of manual labeling and tuning, and greatly improve the system's degree of automation.

   By combining an instrument model and a performance model, and integrating with an automatic and intelligent process of labeling raw data (performance examples and corresponding scores) and constructing models, the framework is able to build a synthesizer for the trumpet, which is known to be among the ones that are most difficult to simulate.

   Compared with other popular synthesis methods, such as sampling synthesis, this new approach provides an intelligent and automatic way of building a music synthesizer that requires many fewer carefully recorded samples, and consumes much less memory. It can work with a very limited number of performance examples and still yield satisfactory synthesis results.

   From a broader perspective, automatically building large complicated pieces of software has always been an important task for Computer Science. For example, in the research field of compilers, a big long-term project was PQCC - production quality compiler com-

piler [34], or building a competitive optimizing compiler from declarative descriptions of language semantics and instruction sets. What we set out to create is the first synthesizer synthesizer - a competitive synthesizer that is created automatically by learning from a limited number of audio recordings and their corresponding scores. This is a perfect example in the continuous effort of conquering one of the grand quests of Computer Science.

In brief, we believe this thesis research has groundbreaking significance in the computer music field.

8.3 Future Work

The synthesis construction process incorporates a lot of domain knowledge and specific instrument characteristics. Still it is general enough that wind instruments other than the trumpet can be easily modeled. A possible direction for future work is to see the framework being applied to different types of musical instruments, as well as various music styles other than classical music, and maybe even expressive performances. It is probably safe to assume that in expressive performances the emotions and variation styles are conveyed in a systematic way [32], and reflected in the amplitude and frequency envelopes, such as the dynamic changes in tempo and loudness. Then the system should be able to learn the "expressiveness" successfully.

Another possible direction of extending the work is to make the system run in real-time. That is, the system will accept high-level control signals triggered by electronic keyboards or other electronic musical instruments, and output corresponding synthetic audio immediately. The current design of the system is for off-line purposes, not because rendering audio takes some time, but mainly due to the fact that it requires contextual information such as slurs. For example, to render a note, the system needs to know the properties (pitch, duration, etc.) of not only the note itself, but also the notes before and after that. But in real-time, one cannot get information about the future, not even the duration of the current note. However, we can probably modify the system to incorporate measures similar to how Synful (see Section 2.1.2.1)

handles real-time MIDI. The quality might suffer a little if the system loses look-ahead ability in real-time. But even a very short time lag, e.g., 50ms, may be enough to at least get duration and slur information based on pitch intervals. So there can be a trade-off between quality and real-timeness. In the case of building a score following application though, we expect the synthesis framework to be fully capable of running in real-time, as the score information is already given.

A crucial measure for evaluating the success of this research is to determine how realistic the synthesized sound is. We use algorithmic metrics, such as squared Euclidean distance, to measure the difference between the actual recording and synthetic one. However, we have always felt that such a metric is rather arbitrary and not as straightforward and convincing as the actual listening tests. Due to resource constraints, we were not able to conduct thorough listening tests. The quality of the constructed synthesis system was merely judged by the author and a few close friends. We expect that in future work listening tests will be essential to back up claims about quality. For now, we invite the readers to listen to sound examples listed in Appendix A.

Appendix A

Sound Examples

The following audio samples are available at `http://www.cs.cmu.edu/~ninghu/thesis/`.

1. Instrument: $B\flat$ Trumpet

   Melody: English Suite Movement V: Finale

   (a) Original audio recording performed by Roger Dannenberg;

   (b) MIDI representing the score;

   (c) Synthesized audio from MIDI using a software synthesizer and with a trumpet instrument sound;

   (d) Synthesized audio from MIDI with the instrument model and performance model described in this thesis. None of the information in (1a) was used to create this example. The models used here were learned from performances of other music;

2. Instrument: Bassoon

   Melody: Emerald - 1st Part

   (a) Original audio recording performed by Mark Dalrymple;

   (b) MIDI representing the score;

(c) Synthesized audio from MIDI using a software synthesizer and with a bassoon instrument sound;

(d) Synthesized audio from MIDI with the instrument model and performance model described in this thesis. None of the information in (2a) was used to create this example. The models used here were learned from performances of other music.

Bibliography

[1] Robert Train Adams. *Electronic Music Composition for Beginners*. Wm. C. Brown, Dubuque, Iowa, 1986. ISBN 0697004570 (pbk.). 6.2

[2] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007. 1

[3] James Beauchamp. Unix workstation software for analysis, graphics, modifications, and synthesis of musical sounds. In *Audio Engineering Society Convention 94*, number 3479, Berlin, March 1993. 4.2.1, 2

[4] Yoshua Bengio. Learning deep architectures for ai. In *Foundations and Trends® in Machine Learning*, volume 2, pages 1–127. Now Publishers Inc., 2009. 2

[5] Alan W. Black. Perfect synthesis for all of the people all of the time. In *IEEE TTS workshop 2002*, 2002. 2.1.1.4

[6] Paul M. Brossier, Juan P. Bello, and Mark D. Plumbley. Fast labelling of notes in music signals. In *ISMIR 2004 Fifth International Conference on Music Information Retrieval Proceedings*, 2004. 4.2.1

[7] MR Celis, JE Dennis, and RA Tapia. A trust region strategy for nonlinear equality constrained optimization. In *Numerical optimization*, volume 1984, pages 71–82. SIAM Philadelphia, 1985. 6.2.1.1

[8] John Chowing. The synthesis of complex audio spectra by means of frequency modulation.

Journal of the Audio Engineering Society, 21(7), 1973. 2.1.1.2

[9] Manfred Clynes. Secrets of life in music: Musicality realised by computer. In *Proceedings of the 1984 International Computer Music Conference*, pages 225–232, San Francisco, CA, 1984. Computer Music Association. 6.2

[10] R. B. Dannenberg and I. Derenyi. Combining instrument and performance models for high-quality music synthesis. *Journal of New Music Research*, (27):211–238, 1998. 1, 2.1.2.2

[11] R B Dannenberg, H Pellerin, and I Derenyi. A study of trumpet envelopes. In *Proceedings of the 1998 International Computer Music Conference*, pages 57–61. International Computer Music Association, 1998. (document), 2.1.2.2, 6, 6.4, 6.5, 6.2, 6.6

[12] R.B. Dannenberg and M. Matsunaga. Automatic capture for spectrum-based instrument models. In *Proceedings of the 1999 International Computer Music Conference. ICMA*, pages 145–148. International Computer Music Association, 1999. 2.1.2.2

[13] Roger B. Dannenberg and Ning Hu. Polyphonic audio matching for score following and intelligent audio editors. In *Proceedings of the 2003 International Computer Music Conference*, pages 27–34, San Francisco, CA, 2003. International Computer Music Association. 4

[14] Roger B. Dannenberg and Sudrit Mohan. Characterizing tempo change in musical performances. In *Proceedings of the 2011 International Computer Music Conference*, pages 650 – 656, San Francisco, CA, August 2011. 6.2.1.1

[15] Istvan Derenyi and Roger B. Dannenberg. Synthesizing trumpet performances. In *Proceedings of the 1998 International Computer Music Conference*, pages 490–496, San Francisco, CA, 1998. International Computer Music Association. 5.1, 1, 5.2

[16] T. Dutoit, V. Pagel, N. Pierret, F. Bataille, and O. Van der Vrecken. The mbrola project: Towards a set of high quality speech synthesizers free of use for non commercial purposes. In *Spoken Language, 1996. ICSLP 96. Proceedings., Fourth International Conference on*,

volume 3, pages 1393 – 1396, 1996. 2.1.1.4

[17] Kelly Fitz. *The Reassigned Bandwidth-Enhanced Method of Additive Synthesis*. PhD thesis, University of Illinois at Urbana-Champaign, 1999. 2.1.1.3

[18] R. Bernard Fitzgerald. *English Suite*. Theodore Presser Company, 2000. Transcribed for Bb Trumpet (or Cornet) and Piano. 4.4.1

[19] Frederick N Fritsch and Ralph E Carlson. Monotone piecewise cubic interpolation. *SIAM Journal on Numerical Analysis*, 17(2):238–246, 1980. 6.2.3

[20] Michael Good. Musicxml for notation and analysis. In *The virtual score: representation, retrieval, restoration*, volume 12, pages 113–124. MIT Press, 2001. 6, 6.1

[21] Hsiao-Wuen Hon, Alex Acero, Xuedong Huang, J. Liu, and M. Plumpe. Automatic generation of synthesis units for trainable text-to-speech systems. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, 1998. 2.2.3

[22] Andrew Horner and Lydia Ayers. Modeling acoustic wind instruments with contiguous group synthesis. *Journal of the Audio Engineering Society*, 46(10):868–879, 1998. 2.1.2.4

[23] Andrew Horner and James Beauchamp. Piecewise-linear approximation of additive synthesis envelopes: A comparison of various methods. In *Computer Music Journal*, volume 20, pages 72–95. MIT Press, Cambridge, MA, 1996. 6.2.1.1

[24] Andrew B. Horner, James W. Beauchamp, and Richard H. Y. So. A search for best error metrics to predict discrimination of original and spectrally altered musical instrument sounds. *Journal of the Audio Engineering Society*, 54(3):140–156, 2006. 6.2

[25] Ning Hu and Roger B. Dannenberg. A comparison of melodic database retrieval techniques using sung queries. In *JCDL 2002: Proceedings of the Second ACM/IEEE-CS Joint Conference on Digital Libraries*, 2002. 4.1.2

[26] Ning Hu and Roger B. Dannenberg. Bootstrap learning for accurate onset detection. *Machine Learning*, 65(2-3):457–471, 2006. 4

[27] Ning Hu, Roger B. Dannenberg, and George Tzanetakis. Polyphonic audio matching and alignment for music retrieval. In *2003 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pages 185–188, New York, 2003. 4, 4.1.1

[28] David Miles Huber. *The MIDI Manual: a Practical Guide to MIDI in the Project Studio*. Focal PressElsevier, 3rd edition edition, 2007. 6.1

[29] Julius O. Smith III. Virtual acoustic musical instruments: Review and update. URL `https://ccrma.stanford.edu/~jos/jnmr/Sampling_Synthesis.html`. 2.1.1.1

[30] Rafael A. Irizarry. The additive sinusoidal plus residual model: A statistical analysis. In *Proceedings of the 1998 International Computer Music Conference*, 1998. 2.1.1.3

[31] Xavier Serra Julius O. Smith III. Parshl: An analysis/synthesis program for non-harmonic sounds based on a sinusoidal representation. In *Proceedings of the 1987 International Computer Music Conference*, 1987. 2.1.1.3

[32] Patrik N Juslin, John A Sloboda, et al. *Music and Emotion*, volume 315. Oxford University Press, New York, 2001. 8.3

[33] Emir Kapanci and Avi Pfeffer. A hierarchical approach to onset detection. In *Proceedings of the 2004 International Computer Music Conference*, pages 438–441, Orlando, 2004. International Computer Music Association. 4

[34] Bruce W Leverett, Roderic GG Cattell, Steven O Hobbs, Joseph M Newcomer, Andrew H Reiner, Bruce R Schatz, and William A Wulf. An overview of the production-quality compiler-compiler project. *Computer*, 13(8):38–49, 1980. 3

[35] Eric Linderman. Music synthesis with reconstructive phase modeling. *Signal Processing Magazine*, 24(2):80 – 91, 3 2007. 2.1.2.1

[36] Lie Lu, Stan Z. Li, and Hong Jiang Zhang. Content-based audio segmentation using support vector machines. In *Proceedings of the IEEE International Conference on Multimedia and*

Expo (ICME 2001), pages 956–959, Tokyo, Japan, 2001. 4

[37] Kantilal Vardichand Mardia, John T. Kent, and John M. Bibby. *Multivariate Analysis*. Probability and Mathematical Statistics. Academic Press, London, 1979. ISBN 0124712525. 6.2.1.2

[38] Matija Marolt, Alenka Kavcic, and Marko Privosnik. Neural networks for note onset detection in piano music. In *Proceedings of the 2002 International Computer Music Conference*. International Computer Music Association, 2002. 4, 4.4.2

[39] R.J. McAulay and Th.F. Quatieri. Speech analysis/synthesis based on a sinusoidal representation. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 34(4):744–754, 1986. 2.1.1.3, 4.2.1, 7.1

[40] Michael Clausen Meinard Muller, Frank Kurth. Audio matching via chroma-based statistical features. In *ISMIR 2005: 6th International Conference on Music Information Retrieval Proceedings*, 2005. 4

[41] Tom M Mitchell. *Machine Learning*. The Mc-Graw-Hill Companies, Inc., Burr Ridge, IL, 1997. 6.2.2

[42] James A. Moorer. How does a computer make music? In *Computer Music Journal*, volume 2, pages 32–37. MIT Press, 1978. 5, 5.1

[43] James Anderson Moorer. Signal processing aspects of computer music: A survey. In *Proceedings of the IEEE*, volume 65, pages 1108–1137. IEEE, 1977. 5.1

[44] Eric Moulines and Francis Charpentier. Pitch-synchronous waveform processing techniques for text-to-speech synthesis using diphones. *Speech Communication*, 9(5–6):453–467, 12 1990. URL `http://www.sciencedirect.com/science/article/pii/016763939090021Z`. 2.1.1.4

[45] P. B. Oncley. Frequency, amplitude, and waveform modulation in the vocal vibrato. *Journal of the Acoustical Society of America*, 49(1A):136 – 136, 1971. 2.2.4.2

[46] Nicola Orio and Diemo Schwarz. Alignment of monophonic and polyphonic music to a score. In *Proceedings of the 2001 International Computer Music Conference*, pages 155–158. International Computer Music Association, 2001. 4

[47] Keiichiro Oura, Ayami Mase, Yoshihiko Nankaku, and Keiichi Tokuda. Pitch adaptive training for hmm-based singing voice synthesis. In *ICASSP 2012: Proceedings of the 2012 International Conference on Acoustics, Speech, and Signal Processing*, 2012. 2.2.4.1

[48] G. De Poli. Audio signal processing by computer. *Music Processing*, (ISBN 0-19-816372-X):73–105, 1993. 1.1, 5

[49] Christopher Raphael. Automatic segmentation of acoustic musical signals using hidden markov model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(4), 1999. 4

[50] Christopher Raphael. A hybrid graphical model for aligning polyphonic audio with musical scores. In *ISMIR 2004: Proceedings of the Fifth International Conference on Music Information Retrieval*, 2004. 4

[51] C. Roads. Physical modeling and formant synthesis. *Computer Music Tutorial*, pages 263–316, 1996. 1.1

[52] Martin Rocamora, Ernesto Lopez, and Luis Jure. Wind instruments synthesis toolbox for generation of music audio signals with labeled partials. In *SBCM09: Proceedings of 2009 Brazilian Symposium on Computer Music*, 2009. 2.1.2.4

[53] Lior Rokach. *Data Mining with Decision Trees: Theory and Applications*, volume 69. World scientific, 2007. 6.2.2

[54] Keijiro Saino, Heiga Zen, Yoshihiko Nankaku, Akinobu Lee, and Keiichi Tokuda. An hmm-based singing voice synthesis system. In *Ninth International Conference on Spoken Language Processing*, 9 2006. 2.2.4.1

[55] Diemo Schwarz. *Data-Driven Concatenative Sound Synthesis*. PhD thesis, Université Paris

6 - Pierre et Marie Curie, 2004. 4

[56] Diemo Schwarz. Current research in concatenative sound synthesis. In *Proceedings of the 2005 International Computer Music Conference*, 2005. 2.1.1.4

[57] Marie-Helene Serra, Dean Rubine, and Roger Dannenberg. Analysis and synthesis of tones by spectral interpolation. *Journal of the Audio Engineering Society*, 38(3):111–128, 1990. URL `http://www.aes.org/e-lib/browse.cfm?elib=6049`. 5

[58] X. Serra. *A System for Sound Analysis/Transformation/Synthesis Based on a Deterministic Plus Stochastic Decomposition*. PhD thesis, Stanford University, 1989. 2.1.1.3, 5

[59] Ferréol Soulez, Xavier Rodet, and Diemo Schwarz. Improving polyphonic and poly-instrumental music to score alignment. In *ISMIR 2003: Proceedings of the Fourth International Conference on Music Information Retrieval*, pages 143–148, Baltimore, 2003. 4

[60] T. Strutz. *Data Fitting and Uncertainty: A Practical Introduction to Weighted Least Squares and Beyond*. Vieweg Verlag, Friedr, & Sohn Verlagsgesellschaft mbH, 2010. ISBN 9783834810229. URL `http://books.google.com.hk/books?id=fudhQgAACAAJ`. 6.2.1.1

[61] Johan Sundberg. Articulatory interpretation of the "singing formant". *Journal of the Acoustical Society of America*, 55(4):838 – 844, 1974. 2.2.4.2

[62] Andrew Swift. A brief introduction to midi. URL `http://www.doc.ic.ac.uk/~nd/surprise_97/journal/vol1/aps2/`. 6.1

[63] Masashi Unoki Takeshi Saitou, Masataka Goto and Masato Akagi. Speech-to-singing synthesis: Converting speaking voices to singing voices by controlling acoustic features unique to singing voices. In *2007 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA 2007) Proceedings*, pages 215–218. IEEE, October 2007. 2.2.4.2, 6.3, 6.3.1, 6.4.4

[64] Tero Tolonen and Matti Karjalainen. A computationally efficient multipitch analysis model. In *IEEE Transactions on Speech and Audio Processing*, volume 8, pages 708–716. IEEE, 2000. 7.1

[65] Visiv. Sharpeye: Music scanning, 2008. URL `http://www.visiv.co.uk/`. 6.1

[66] D Wessel, C Drame, and M Wright. Removing the time axis from spectral model analysis-based additive synthesis: Neural networks versus memory-based machine learning. In *Proceedings of the 1998 International Computer Music Conference*, pages 62–65, San Francisco, CA, 1998. International Computer Music Association. 2.1.2.3, 5.1.1

[67] Wikipedia. Wikipedia - additive synthesis, . URL `http://en.wikipedia.org/wiki/Additive_synthesis`. 2.1.1.3

[68] Wikipedia. Wikipedia - concatenative synthesis, . URL `http://en.wikipedia.org/wiki/Concatenative_synthesis`. 2.1.1.4

[69] Wikipedia. Wikipedia - sample based synthesis, . URL `http://en.wikipedia.org/wiki/Sample-based_synthesis`. 2.1.1.1

[70] H. Zen, K. Tokuda, T. Masuko, T. Kobayashi, and T. Kitamura. Hidden semi-markov model based speech synthesis. In *ICSLP 2004: Proceedings of the 2004 International Conference on Spoken Language Processing*. IEEE, 2004. 2.2.3, 2.2.4.1

[71] Aymeric Zils and Francois Pachet. Musical mosaicing. In *Proceedings of the COST G-6 Conference on Digital Audio Effects*, 2001. 2.1.1.4